Does QUIC make the Web faster ?

Prasenjeet Biswal
Department of Computer Science
University of Houston

Omprakash Gnawali
Department of Computer Science
University of Houston

***Abstract*—Increase in size and complexity of web pages has challenged the efficiency of HTTP. Recent developments to speed up the web have resulted in two promising protocols, HTTP/2 (RFC 7540) at the application layer and QUIC (multiplexed stream transport over UDP). Google servers are using HTTP/2 and QUIC whereas other major sites like Facebook and Twitter have begun using HTTP/2. In this paper, we compare the performance of HTTP/2 vs QUIC+SPDY 3.1 by studying the Web page load times. In the first set of experiments, we serve synthetic pages (only static objects) over both protocols in emulated controlled network conditions and then extend it to real network, both wired and cellular (2G/3G in India and 3G/4GLTE in US). Further, we conduct experiments on a set of web pages on the most popular sites from the Internet (Alexa Rankings) in controlled conditions. We find QUIC to perform better overall under poor network conditions (low bandwidth, high latency and high loss), for e.g. more than 90% of synthetic pages loaded faster with QUIC in 2G compared to 60% in 4GLTE. This is due to the lower connection establishment latency and improved congestion control mechanism in QUIC. However, QUIC does not offer significant advantage when a webpage consists of many small-sized objects.**

1. INTRODUCTION

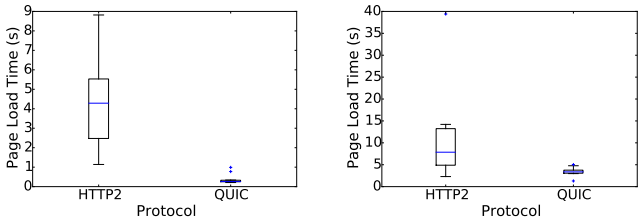
HTTP/1.1 is an application layer protocol, introduced in 1997 to transfer simple HTML pages over the Internet. The size and complexity of pages have increased multi-fold since then [1], [2]. While several modifications in protocols (like HTTP pipelining, TCP Fast Open), algorithms (data compression techniques), browser implementations and faster link speeds have helped to speed up the web, the base protocol has resisted efforts for a major upgrade until very recently.

In 2012, Google proposed SPDY as an enhancement to HTTP/1.1 [3]. HTTP/2, which is largely based on SPDY, was standardized and published as RFC 7540 in May 2015. HTTP/2 provides several improvements over HTTP/1.1 like (1) multiplexed streams over a single TCP connection, (2) binary message framing for more efficient message processing, (3) header compression (HTTP/2 uses HPACK (RFC 7541) [4] compression), (4) server push thus eliminating the latency due to round trips due to client request, and (5) request prioritization.

HTTP/2 still has some performance limitations in the way it is used: head-of-line (HOL) blocking at the client due to TCP's in-order delivery and larger connection establishment time due to TCP's 3-RTT handshake (with TLS). Studies suggest that increasing the bandwidth past a point ($\sim$ 5 Mbps) does not reduce the page load time as the decrease in delay can [5]. QUIC (Quick UDP Internet Connections) was proposed in 2013 by Google to address these problems [6], [7]. QUIC's key features include (1) reduced connection establishment time (0-RTT) with security comparable to HTTPS, (2) improved congestion control - TCP Cubic + Packet Pacing with Forward Error Correction (FEC) packets (currently not enabled) to reduce retransmissions (but [8] indicates that FEC results in poor link utilization), (3) Connection Migration - QUIC uses connection identifier (CID) to uniquely identify a connection and eliminates reconnections in mobile clients during switch-over. QUIC promises improved multiplexing than HTTP/2 (which also solves head-of-line (HOL) blocking).

There has been interest in making QUIC part of the future standardization, thus studying its performance is important not only for understanding how protocol mechanisms perform in different circumstances but also to inform the Internet standardization bodies. This study informs the Internet community about the performance of QUIC so its design can evolve. Due to similarity in some mechanisms used by HTTP/2 and QUIC and the interest in borrowing mechanism [9], the insights from this study may also be applicable to HTTP/2. The study will also help web designers, network administrators, server and client engineers to design pages or systems or tweak network configurations to best utilize the features of QUIC and HTTP/2.

In this work, we compare the performance of QUIC and HTTP/2 and explain the reason for those differences. The measurement study is not trivial. The only QUIC server available is a sample server provided by Google, a basic in-memory server, and features like FEC are not implemented. Only a few browsers (Google Chrome, Chromium and Opera) support both HTTP/2 and QUIC. The page load time for the a given webpage can be highly variable regardless of the server, especially under high loss (See Figure 1). The variations become more prominent when the page consists of scripts and stylesheets. Further, the measurements for



(a) High Loss, Low Delay (b) High Loss, High Delay

Figure 1: High variance in Page Load Time under high loss, especially for HTTP/2.

complete web sites are specifically representative of network conditions at that time and the content of webpages. The results with these pages cannot be reproduced at different instances of time and for different users, because the number and size of embedded objects which are crucial in the performance study change rapidly and are personalized for sites like **cnn.com** or **amazon.com**. Therefore, we had to design web pages consisting of different combinations of object sizes and numbers for our experiments in controlled conditions.

We next review related work (§2) followed by extensive study of QUIC and HTTP/2's performance in different scenarios(§3). Finally we conclude and list the limitations of this work (§4). Throughout the paper, QUIC refers to QUIC as transport layer and SPDY as the application layer. Similarly, HTTP/2 means HTTP/2 as application layer over TCP as transport layer and SPDY means SPDY as application layer over TCP as transport layer.

2. RELATED WORK

Studies	HTTP	SPDY	HTTP/2	QUIC
[8], [10]	X	X		X
[11], [12]		X	X	
[13], [14]	X	X		
Our Study			X	X

TABLE 1: Previous studies comparing HTTP, SPDY, QUIC and HTTP/2.

QUIC studies: Although QUIC is relatively new, there have been few performance studies with QUIC. Carlucci et. al. [8] studied the impact of QUIC on goodput, link utilization and compared page load time for QUIC, SPDY and HTTP. The authors find that QUIC has better link utilization than HTTP (TCP), especially under high loss. Further, QUIC overall loads pages faster than HTTP and outperforms SPDY in high loss. However, FEC when enabled does not provide much benefit and reduces goodput (FEC packets use 33% of available bandwidth). In our work, we account for larger parameter space both for network conditions and composition of synthetic pages and use QUIC version 23 (compared to version 21) which increases the number of parallel streams from 6 to 110.

Somak Das studied page load times of Alexa's Top 500 sites [15] over QUIC, SPDY and HTTP/1.1 under different network configurations of bandwidth and delay [10]. The author reports that HTTP outperforms QUIC which outperforms SPDY on "very low bandwidth (0.2 Mbps)" link, but for low bandwidth links (0.3 - 1 Mbps) QUIC is better than HTTP/1.1. QUIC improves as RTT increases but is slower on high bandwidth and low-RTT scenarios. QUIC loads HTTPS sites much faster than HTTP/1.1. For mobile web, experiments are performed over cellular network traces and an enhanced congestion control protocol (Sprout-EWMA) is suggested. In our study, we additionally inject loss into the network to further study QUIC's performance in challenging environments. Also, we do an exhaustive study of performance on synthetic pages, which provides a better evaluation due to minimal dependency (among objects) and computation (stylesheets, scripts) compared to Alexa sites. We also perform comparisons across cellular networks spanning different technologies (2G, 3G, 4GLTE) and geographies (the US and India).

Unlike previous studies, we compare QUIC against HTTP/2, the new Internet Web standard (and likely to become a dominant protocol in the near future) and successor of SPDY (see the section on HTTP/2 vs SPDY below).

Google claims about 50% of the requests from Chrome to Google servers are handled over QUIC [16]. It reports QUIC works better than TCP on low-bandwidth connections, reducing the page load time for Google home page almost by a second on 1% of slowest connections. Over 75% connections have benefited from 0-RTT; even optimized websites have a 3% improvement in page load time. According to Google, benefits are more prominent in video services like YouTube (User feedbacks report 30% less buffering).

SPDY Studies: Wang et al. compare HTTP/1.1 and SPDY [14]. They use Apache's `mod_spdy` module for SPDY and regular Apache for HTTP/1.1. The initial experiments consist of serving synthetic pages with SPDY and HTTP under different network conditions. They found that SPDY is better for small objects and many large objects under low loss. They state that the main contributor to SPDY's better performance is due to its multiplexing over single TCP connection. This reduces total connection establishment time and results in fewer retransmissions (concurrent connections in HTTP/1.1 compete and induce loss), but the same feature prove detrimental under high loss due to aggressive congestion control mechanism (In HTTP/1.1, only 1 of 6 TCP connections is affected). Further they conduct experiments for real web pages (Alexa's [15] top 200 websites) using a self-designed module Epload to emulate page loads to get rid of browser dependencies and computations. They found that SPDY loads 70% of the websites faster over different network conditions.

A study has suggested that SPDY is not very beneficial over HTTPS and HTTP [13]. Their results suggest that SPDY is only 4.5% faster than HTTPS and 3.4% slower than HTTP connections. Also the median acceleration of SPDY over HTTPS was only 1.9%. They identify the main reason for low improvement of SPDY over HTTP(S) to be domain

Parameters	Range	High
bandwidth	1Mbps, 100Mbps	100Mbps
round trip time	10ms, 100ms, 200ms	≥100ms
packet loss (%)	0, 1, 2	≥1
object size	100B, 1K,10K, 100K	≥10K
#objects	2, 8, 16, 64, 128	≥64

TABLE 2: Factors for controlled experiments.

Type	Freq.	Core	Mem.	OS	Kernel
Client	1.2 Ghz	8	8 GB	14.04 (64-bit)	3.13.0-45
Server A (Controlled)	800 Mhz	2	4 GB	14.04 (64-bit)	3.13.0-44
Server B (Uncontrolled)	2.66 Ghz	4	8 GB	14.04 (64-bit)	3.13.0-44

TABLE 3: Server and Client Machine characteristics

sharding (SPDY works better for a single host website but most real-world websites download resources from different domains) and loading dependency of the web resources. Negative effects of domain sharding on SPDY performance is also discussed in [17].

Zaki et al. [18] compare SPDY with HTTP over 42 websites in Accra, Abu Dhabi, Bremen and New York. They report that SPDY performs better than HTTP in low-bandwidth networks (e.g. Accra).

HTTP/2 vs. SPDY: A comparison between raw HTTPS, HTTP/2 and SPDY/3.1 is discussed in [11]. They used HttpWatch [19] with Firefox to run simple page load tests on Google's UK homepage (https://www.google.co.uk/). They find that HTTP/2 packets have significantly small header size compared to SPDY because of HPACK header compression in HTTP/2. SPDY in turn has smaller header compared to HTTPS (HTTPS does not use header compression). The response body size of HTTPS is largest followed by HTTP/2 followed by SPDY. The larger response body of HTTP/2 compared to SPDY is a result of padding in data frames for greater security. The number of connections established are same for HTTP/2 and SPDY, in both cases fewer than with HTTPS. The page loads with HTTP/2 is faster than SPDY while HTTPS lags far behind due to larger connection establishment time. HTTP/2 outperforms SPDY due to smaller GET requests (HPACK compression) over asymmetric links. Other than the pointers discussed in [11], [12] states that HTTP/2 also benefits from multi-host multiplexing, improved prioritization and faster encrypted connections (ALPN extension instead of NPN).

To the best of our knowledge, this paper is the first work comparing QUIC with HTTP/2.

3. PERFORMANCE EVALUATION

3.1. Experimental Setup

We compare QUIC and HTTP/2 under controlled and uncontrolled environments. Table 2 provides the complete list of web page composition and network conditions for controlled experiments. To emulate different network

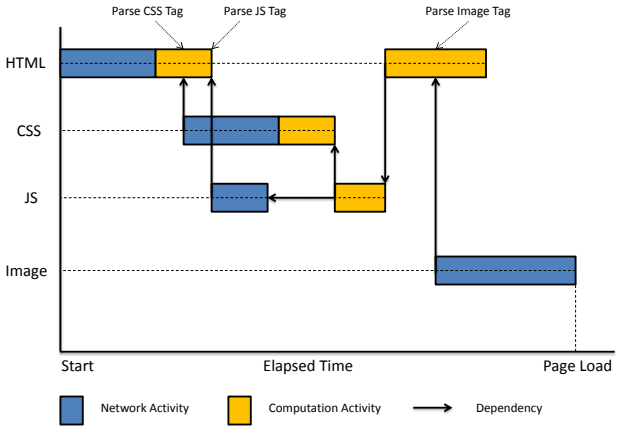
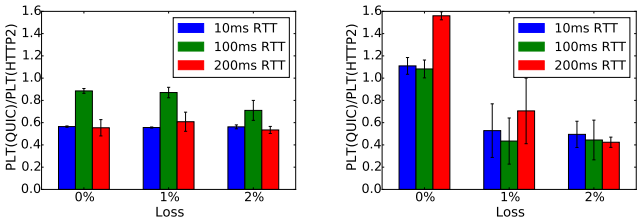


Figure 2: Different steps in a typical web page load. Reproduced partly from [14].



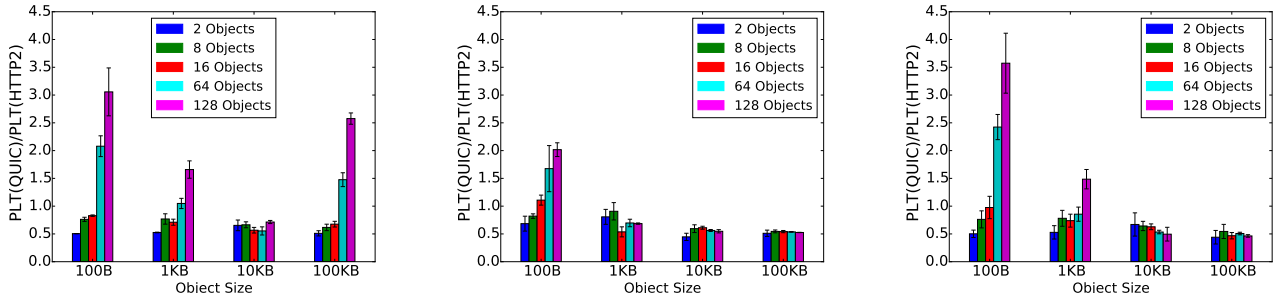
(a) 1 Mbps, 10KB, 64 Objects (b) 100Mbps, 10KB, 64 Objects

Figure 4: PLT(QUIC)/PLT(HTTP/2) for different combinations of loss and RTT.

conditions, we use Netem [20].

We use QUIC server (version 23) provided by Google. OpenLiteSpeed (version 1.3.9) [21] is used for handling HTTP/2 (draft-17) requests. We select OpenLiteSpeed because its HTTP/2 server is written in C++ (same as QUIC toy server) and supports the latest HTTP/2 draft-17. For each page load, we perform the experiment five times and take the median value to account for variance in measurements. Table 3 lists the client and server machine configurations. For controlled experiments, Server A and client machine were connected through a router. For uncontrolled experiments (both wired and mobile networks), a publicly accessible Server B was used. Chromium Browser v41.0.2272.76 is used as client for all experiments. Our goal is to compare the protocol performance within (not across) controlled and uncontrolled environment, so the difference in characteristics of the two servers does not impact the conclusions from this study.

To measure page load time, we developed an extension for Chromium client. The extension measures the total time spent solely in network activity by adding the time difference between the receipt of last byte and queuing time of all resources. We consider longest interval if loading time of



(a) Conf-3 (latency=200ms, loss=0%) (b) Conf-7 (latency=10ms, loss=2) (c) Conf-9 (latency=200ms, loss=2%)

Figure 3: PLT(QUIC)/PLT(HTTP/2) for varying object sizes and number of objects

Configuration	1	2	3	4	5	6	7	8	9
Loss%	0	0	0	1	1	1	2	2	2
RTT (ms)	10	100	200	10	100	200	10	100	200

TABLE 4: The nine configurations used in the controlled environment experiments comparing the protocols.

multiple resources overlap. The extension only sums up the time taken in loading resources and discards the time taken for script execution. This helps to mitigate the variance in page load time to some extent. Fig. 2 depicts a typical download of a web page. The page load time (PLT) in this scenario is the time spent to download HTML, CSS and Image; JS download time does not contribute to the total page load time in this example. PLT also does not include the time taken to execute CSS, JS and some portion of HTML. The extension issues a xmlHTTPRequest to the Apache server running on same machine (client) which writes the measurements to a file. **Metric:**. We use the performance metric PLT(QUIC)/PLT(HTTP/2) to compare the protocols, where PLT stands for Page Load Time. If the page load is faster with QUIC than with HTTP/2, the ratio will be smaller than 1.0.

3.2. Synthetic Pages over Controlled Environment

We conduct experiments over the complete parameter space provided in Table 2 and the configurations listed in Table 4. We organize the main results from the performance comparison so we can understand the impact of these two factors on the page load times:

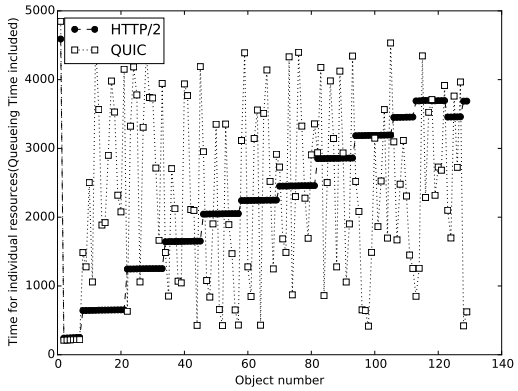
Impact of Page Content on QUIC's Performance: The number of objects and the size of objects affect the performance of protocols. Modern webpages have evolved to not only contain a range of object sizes but also a large range in terms of number of objects per page. Hence, we evaluate the protocols over a range of object sizes and the number of objects. Figs. 3(a), (b) and (c) show that as the number of objects increase (especially if the object size is small), HTTP/2 starts to outperform QUIC in terms of speed. However QUIC improves relatively over HTTP/2 as objects become larger and loss is introduced (see figs. 3 (b) and 3 (c)). The most suitable explanation for this condition is the head of line blocking experienced by HTTP/2 plus and efficient congestion control in QUIC.

Impact of Network Characteristics on QUIC's Performance: QUIC is designed for fast connection establishment and combating lossy networks. Consider the figure 4(b). The first group represents perfect conditions for HTTP/2 - high bandwidth, 0 loss. As expected HTTP/2 performs better than QUIC. But when loss is introduced, QUIC performs better than HTTP/2 as shown in the second and third group of bar plots. One question arises - Why does the performance of QUIC not improve when RTT is increased to 200ms (first group) even though QUIC has 0-RTT connection establishment? Here is a possible explanation - Though the 0-RTT connection establishment adds to QUIC's performance, it is prominent only on pages with a few small-sized objects (See graphs for 2, 8 and 16 objects in figs. 3 (a), 3 (b) and 3 (c)). On pages with large-sized objects, time spent in establishing connection is a tiny fraction of the time in loading the objects. QUIC performs better than HTTP/2 in lower bandwidth conditions. In contrast to figure 4(b), in figure 4(a), even the first bar group representing no network loss is less than 1.

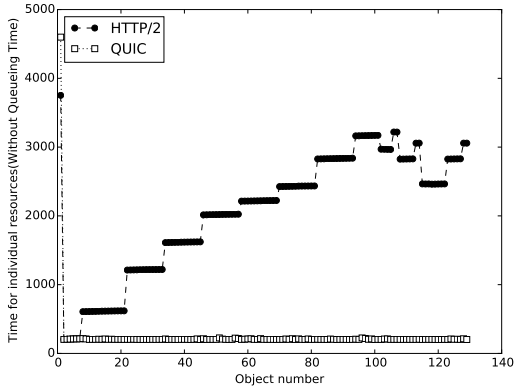
Summary of results over the complete parameter space: Through extensive experiments, we find that QUIC beats HTTP/2 under the following conditions:

1) the bandwidth decreases. QUIC is faster in 78% cases for 100Mbps link compared to 85% for 1Mbps. This is a marginal victory for QUIC as compared to other scenarios.
2) the size of the objects increase. For pages consisting of large objects (10KB and 100KB), QUIC performs better in 96% cases (95% for 100Mbps connection and 97.7% for 1Mbps connection.)
3) loss is injected into the network. Overall QUIC is faster in 82% of cases when loss is introduced (80% for 100 Mbps and 85% for 1 Mbps connection).

HTTP/2 loads a page faster when page consists of a large number of small-sized objects and there is minimal loss ($\sim$0%) in the network. Logically this condition should have been ideal for QUIC as this page composition can exploit



(a) Queuing Time included



(b) Queuing Time not included

Figure 5: Load time of resources. Configuration : 1Mbps, #Objects 128, Object Size 1KB, loss 0%, delay 200ms

the far more efficient multiplexing available in QUIC. To understand this contradictory behavior, we conducted experiments with web page consisting of 128 objects of one kilobyte under 1Mbps connection with a delay of 200ms and loss of 0%. As can be seen from figures 5 (a) and (b), for QUIC, the actual resource load time (time between request of resource to receipt of last byte) is very small compared to HTTP/2, whereas the total time to fetch each object once it is parsed in html is larger for many objects. One possible reason for this is that the QUIC toy server is not optimized to be used efficiently with Chromium browser.

To summarize, QUIC is faster (1.2-4.5X) when pages consist of few small-sized objects. QUIC also performs better when the page consists of large-sized objects. QUIC can achieve faster page loads in poor network conditions of lower bandwidth, high delay and lossy network. However, Page load with QUIC is slower (1.1-3.2X) when page consists of many small-sized objects.

3.3. Synthetic Pages over Wired Networks

We repeat the experiments over wired network on UH Campus LAN. HTTP/2 and QUIC servers run on publicly accessible machine. The difference from controlled environment is that bandwidth, loss and delay in this case are

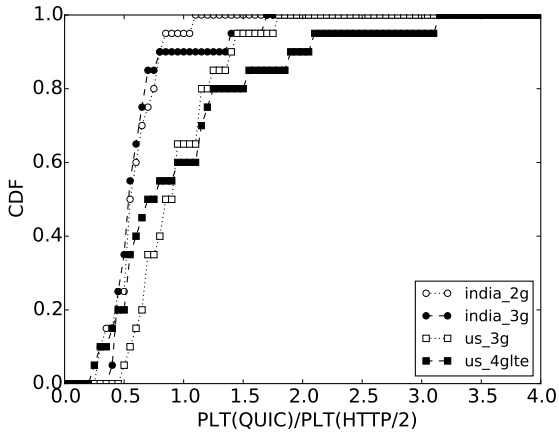


Figure 6: Performance of synthetic pages with QUIC and HTTP/2 under different cellular networks with clients in India and the US.

what is available on the live campus network. During the experiments, the network had a Median Bandwidth of 51 Mbps, Median Delay of 31 ms, and Median Loss of 0%. We find that the results from these experiments are similar to the results from controlled experiments under configurations 1 and 2 (0 loss and 10ms and 100ms latency).

3.4. Synthetic Pages over Cellular networks in the US and India

We run experiments with four different settings:

1) Client runs in India with Airtel 2G data enabled. We observed a network datarate of 170 Kbps and a delay of 334 ms.
2) Client runs in India with Airtel 3G data enabled. We observed a network datarate of 7.78 Mbps and a delay of 153 ms.
3) Client runs in the US with TMobile 3G data enabled. We observed a network datarate of 3.54 Mbps, loss of 0.048%, and a delay of 94ms.
4) Client runs in the US with TMobile 4GLTE data enabled. We observed a network datarate of 19.16 Mbps, loss of 0%, and a delay of 32ms

Figure 6 shows that more than 90% of pages have lower page load times with QUIC than with HTTP/2 when accessed from India. Compared to this only 60% of pages are faster over QUIC when client is in the US (low delay, high bandwidth, low loss). Also QUIC performs better in poor network conditions (2G in India and 3G in US) compared to their higher-speed counterparts (3G in India and 4GLTE in the US). This reaffirms, using real-world studies in two different countries, the earlier result of QUIC performing better in high RTT and lossy network.

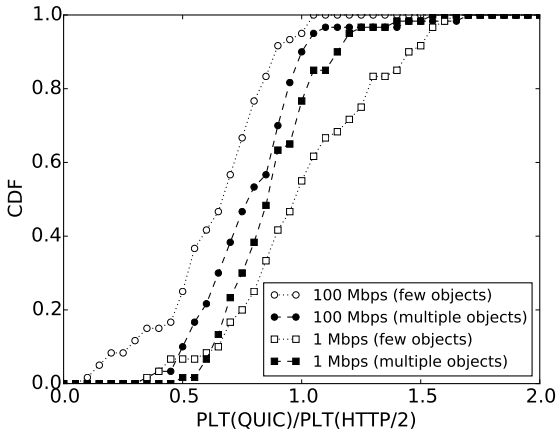


Figure 7: Performance of QUIC for complete web sites.

3.5. Complete Websites over Controlled Environment

Finally, we conduct our experiments on complex web pages consisting of images, stylesheets and scripts. We download top 200 sites from Alexa top sites [15]. From within this set, we selected 15 sites with small ($\leq$20) and other 15 with large (80-100) number of objects. Experiments are performed over four configurations (1, 3, 7 and 9 as shown in Table 4) with loss and delay varying from low to high.

Figure 7 shows that the maximum value of PLT(QUIC)/PLT(HTTP/2) drops to 1.6 from 3.2 (in synthetic pages). This is because the script execution time effectively serializes the loading of multiple objects on a page as the objects to be fetched are computed by the script. Also contrary to assumption, even dense pages (large number of objects) load faster over QUIC. The reason for this is that average page size is 2.5 MB and average object size is 25.39 KB (QUIC works better with large objects).

4. CONCLUSIONS

QUIC (stream multiplexed transport over UDP) was developed to speed up Web page loading, features 0-RTT connection, improved congestion control and multiplexing without head-of-line (HOL) blocking. Our experiments over controlled environment suggest that QUIC loads pages quicker than HTTP/2 under poor network conditions characterized by low-bandwidth, high delay and high loss. QUIC performance also improves with increase in object size. Increase in the number of embedded objects impacts QUIC's performance due to inefficient browser queuing. Our experiments over uncontrolled conditions (wired and cellular networks in the US and in India) confirm these observations. Experiments over complete web sites show performance gain with QUIC even for large number of embedded objects due to their large size.

Limitations: First, the only available QUIC server, which we used in this study, is not production-ready compared to OpenLiteSpeed's HTTP/2. Second, we do not quantitatively evaluate the effect of dependencies and computation on loading of embedded objects for complete web sites. Third, we did not conduct experiments on dynamic pages that are dependent on server processing.

References

[1] "Average Web Page breaks 1600k (jul. 18th 2014)," http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[2] T. Everts, "Web Performance Today (Jun. 5th 2013)," http://www.webperformancetoday.com/2013/06/05/web-page-growth-2010-2013/.

[3] "SPDY Protocol Draft." https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00.

[4] "HPACK: Header Compression for HTTP/2," https://tools.ietf.org/html/rfc7541.

[5] I. Grigorik, "Latency: The New Web Performance Bottleneck," https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/.

[6] "QUIC: Design Document and Specification Rationale." https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/mobilebasic?pli=1.

[7] "Quic Wire Layout Specification." https://docs.google.com/document/d/1WJvyZflAO2pq77yOLbp9NsGjC1CHetAXV8I0fQe-B_U/edit?pref=2&pli=1.

[8] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," 2015.

[9] "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2 draft-tsvwg-quic-protocol-02," https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02.

[10] S. R. Das, "Evaluation of QUIC on Web Page Performance," Master's thesis, Massachusetts Institute of Technology, 2014.

[11] "A Simple Performance Comparison of HTTPS, SPDY and HTTP/2 (16th Jan. 2015)," https://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/.

[12] J. Dorfman, "The Shift from SPDY to HTTP/2 (Mar. 2nd 2015)," https://www.maxcdn.com/blog/spdy-http2-shift/.

[13] G. Podjarny, "Not as SPDY as You Thought (Jun. 12th 2012)," http://www.guypo.com/not-as-spdy-as-you-thought/.

[14] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 387–399.

[15] "Alexa : The Web Information Company." http://www.alexa.com/topsites/countries/US., 2015.

[16] "A QUIC update on Google's experimental transport." http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html.

[17] J. Graham, "Using CloudFlare to mix domain sharding and SPDY (26th Dec. 2013)," https://blog.cloudflare.com/using-cloudflare-to-mix-domain-sharding-and-spdy/.

[18] Y. Zaki, J. Chen, T. Pötsch, T. Ahmad, and L. Subramanian, "Dissecting Web Latency in Ghana," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 241–248.

[19] "HTTPWatch," http://www.httpwatch.com/.

[20] "NeteM : Linux Network Emulator," http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[21] "OpenLiteSpeed." http://open.litespeedtech.com/mediawiki/.

Can You See Me Now? A Measurement Study of Zoom, Webex, and Meet

Hyunseok Chang
Nokia Bell Labs
Murray Hill, NJ, USA

Matteo Varvello
Nokia Bell Labs
Murray Hill, NJ, USA

Fang Hao
Nokia Bell Labs
Murray Hill, NJ, USA

Sarit Mukherjee
Nokia Bell Labs
Murray Hill, NJ, USA

ABSTRACT

Since the outbreak of the COVID-19 pandemic, videoconferencing has become the default mode of communication in our daily lives at homes, workplaces and schools, and it is likely to remain an important part of our lives in the post-pandemic world. Despite its significance, there has not been any systematic study characterizing the user-perceived performance of existing videoconferencing systems other than anecdotal reports. In this paper, we present a detailed measurement study that compares three major videoconferencing systems: Zoom, Webex and Google Meet. Our study is based on 48 hours' worth of more than 700 videoconferencing sessions, which were created with a mix of emulated videoconferencing clients deployed in the cloud, as well as real mobile devices running from a residential network. We find that the existing videoconferencing systems vary in terms of geographic scope, which in turns determines streaming lag experienced by users. We also observe that streaming rate can change under different conditions (e.g., number of users in a session, mobile device status, etc), which affects user-perceived streaming quality. Beyond these findings, our measurement methodology can enable reproducible benchmark analysis for any types of comparative or longitudinal study on available videoconferencing systems.

CCS CONCEPTS

• **Networks** → **Network measurement**; *Cloud computing*; • **Information systems** → *Collaborative and social computing systems and tools*.

ACM Reference Format:
Hyunseok Chang, Matteo Varvello, Fang Hao, and Sarit Mukherjee. 2021. Can You See Me Now? A Measurement Study of Zoom, Webex, and Meet. In *ACM Internet Measurement Conference (IMC '21), November 2–4, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3487552.3487847

IMC '21, November 2–4, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9129-0/21/11…$15.00
https://doi.org/10.1145/3487552.3487847

1 INTRODUCTION

There is no doubt that the outbreak of the COVID-19 pandemic has fundamentally changed our daily lives. Especially with everyone expected to practice physical distancing to stop the spread of the pandemic, various online communication tools have substituted virtually all sorts of in-person interactions. As the closest form of live face-to-face communication in a pre-pandemic world, videoconferencing has practically become the default mode of communication (e.g., an order-of-magnitude increase in videoconferencing traffic at the height of the pandemic [23, 28, 37]). Thanks to its effectiveness and reliability, video communication is likely to remain an important part of our lives even in the post-pandemic world [27, 32].

Despite the critical role played by existing videoconferencing systems in our day-to-day communication, there has not been any systematic study on quantifying their performance and Quality of Experience (QoE). There is no shortage of anecdotal reports and discussions in terms of the usability, video quality, security, and client resource usage of individual systems. To the best of our knowledge, however, no scientific paper has yet investigated the topic thoroughly with a sound measurement methodology that is applicable across multiple available systems. Our main contribution in this paper addresses this shortcoming.

In this paper, we shed some light on the existing videoconferencing ecosystems by characterizing their infrastructures as well as their performance from a user's QoE perspective. To this end, we have devised a measurement methodology which allows us to perform controlled and reproducible benchmarking of videoconferencing systems by leveraging a mix of emulated videoconferencing clients deployed in the cloud, as well as real mobile devices running from a residential network. We provide the detailed description of our methodology as well as the open-source tools we used (Sections 3 and 4), so that anyone can replicate our testbed to repeat or further extend our benchmark scenarios. Driven by our methodology, we investigate three popular videoconferencing systems on the market: Zoom, Webex and Google Meet (Meet for short). Each of these platforms provides a free-tier plan as well as paid subscriptions, but we focus on their *free-tier plans* in our evaluation. Given these three systems, we conduct measurement experiments which take a combined total of 48 videoconferencing hours over more than 700 sessions, with 200 VM hours rented from 12 geographic locations and 18 hours of two Android phones hooked up at one location. Our findings include:

Finding-1. In the US, typical streaming lag experienced by users is 20–50 ms for Zoom, 10–70 ms for Webex, and 40–70 ms for Meet. This lag largely reflects the geographic separation of users (e.g., US-east vs. US-west). In case of Webex, all sessions created in the US appear to be relayed via its infrastructure in US-east. This causes the sessions among users in US-west to be subject to artificial detour, inflating their streaming lag.

Finding-2. Zoom and Webex are characterized by a US-based infrastructure. It follows that sessions created in Europe experience higher lag than those created in the US (90–150 ms for Zoom, and 75–90 ms for Webex). On the other hand, the sessions created in Europe on Meet exhibit smaller lag (30–40 ms) due to its cross-continental presence including Europe.

Finding-3. All three systems appear to optimize their streaming for low-motion videos (e.g., a single-person view with a stationary background). Thus high-motion video feeds (e.g., dynamic scenes in outdoor environments) experience non-negligible QoE degradation compared to typical low-motion video streaming.

Finding-4. Given the same camera resolution, Webex sessions exhibit the highest traffic rate for multi-user sessions. Meet exhibits the most dynamic rate changes across different sessions, while Webex maintains virtually constant rate across sessions.

Finding-5. Videoconferencing is an expensive task for mobile devices, requiring at least 2–3 full cores to work properly. Meet is the most bandwidth-hungry client, consuming up to one GB per hour, compared to Zoom's gallery view that only requires 175 MB per hour. We estimate that one hour's videoconferencing can drain up to 40% of a low-end phone's battery, which can be reduced to about 20–30% by turning off the onboard camera/screen and relying only on audio. All videoconferencing clients scale well with the number of call participants, thanks to their UI which only displays a maximum of four users at a time.

The rest of the paper is organized as follows. We start by introducing related works in Section 2. We then present our measurement methodology in Section 3, and describe the measurement experiments and our findings in detail in Sections 4–5. We conclude in Section 6 by discussing several research issues.

2 RELATED WORK

Despite the prevalence of commercial videoconferencing systems [36], no previous work has directly compared them with respect to their infrastructures and end-user QoE, which is the main objective of this paper. The recent works by [34] and [29] investigate the network utilization and bandwidth sharing behavior of existing commercial videoconferencing systems based on controlled network conditions and client settings. Several works propose generic solutions to improve videoconferencing. For example, Dejavu [25] offers up to 30% bandwidth reduction, with no impact on QoE, by leveraging the fact that recurring videoconferencing sessions have lots of similar content, which can be cached and re-used across sessions. Salsify [24] relies on tight integration between a video codec and a network transport protocol to dynamically adjust video encodings to changing network conditions.

Videoconferencing system	Low quality	High quality
Zoom [7]	600 Kbps	
Webex [8]	500 Kbps	2.5 Mbps
Meet [14]	1 Mbps	2.6 Mbps

Table 1: Minimum bandwidth requirements for one-on-one calls.

As a consequence of the COVID-19 pandemic, the research community has paid more attention to the impact of videoconferencing systems on the quality of education [21, 33, 40]. As educational studies, these works rely on usability analysis and student surveys. In contrast, our work characterizes QoE performance of the videoconferencing systems using purely objective metrics.

Videoconferencing operators do not provide much information about their system, e.g., footprint and encoding strategies. One common information reported by each operator is the minimum bandwidth requirements for one-on-one calls (Table 1). The results from our study are not only consistent with these requirements, but also cover more general scenarios such as multi-party sessions.

3 BENCHMARKING DESIGN

In this section, we describe the benchmarking tool we have designed to study existing commercial videoconferencing systems. We highlight key design goals for the tool first, followed by associated challenges, and then describe how we tackle the challenges in our design.

A videoconferencing system is meant to be used by end-users in mobile or desktop environments that are equipped with a camera and a microphone. When we set out to design our benchmarking tool for such systems, we identify the following design goals.

(D1) Platform compliance: We want to run our benchmark tests using *unmodified* official videoconferencing clients with *full audiovisual capabilities*, so that we do not introduce any artifact in our evaluation that would be caused by client-side incompatibility or deficiency.

(D2) Geo-distributed deployment: To evaluate web-scale videoconferencing services in realistic settings, we want to collect data from *geographically-distributed* clients.

(D3) Reproducibility: We want to leverage *a controlled, reproducible client-side environment*, so that we can compare available videoconferencing systems side-by-side.

(D4) Unified evaluation metrics: We want to evaluate different videoconferencing platforms based on *unified metrics* that are applicable across all the platforms.

It turns out that designing a benchmarking tool that meets all the stated goals is challenging because some of these goals

are in fact conflicting. For example, while geographically-dispersed public clouds can provide distributed vantage point environments (**D2**), cloud deployments will not be equipped with necessary sensory hardware (~~**D1**~~). Crowd-sourced end-users can feed necessary audiovisual data into the videoconferencing systems (**D1**), but benchmarking the systems based on unpredictable human behavior and noisy sensory data will not give us objective and reproducible comparison results (~~**D3**~~). On the other hand, some goals such as **(D3)** and **(D4)** go hand in hand. Unified evaluation metrics alone are not sufficient for comparative analysis if reproducibility of client-side environments is not guaranteed. At the same time, reproducibility would not help much without platform-agnostic evaluation metrics.

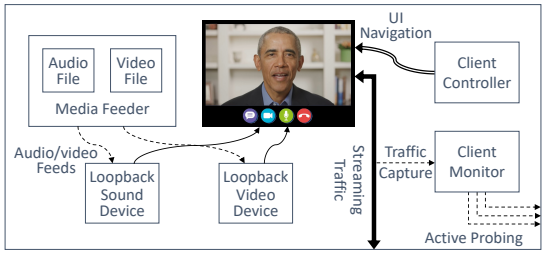


Figure 1: Cloud VM in a fully-emulated environment.

3.1 Design Approach

Faced with the aforementioned design goals and challenges, we come up with a videoconferencing benchmark tool that is driven by three main ideas: (i) client emulation, (ii) coordinated client deployments, and (iii) platform-agnostic data collection.

Client emulation. One way to circumvent the requirement for sensory devices in videoconferencing clients is to *emulate* them. Device emulation also means that the sensory input to a videoconferencing client would be completely under our control, which is essential to reproducible and automated benchmarking. To this end, we leverage *loopback pseudo devices* for audio/video input/output. In Linux, `snd-aloop` and `v4l2loopback` modules allow one to set up a virtual soundcard device and a virtual video device, respectively. Once activated these loopback devices appear to videoconferencing clients as standard audio/video devices, except that audiovisual data is not coming from a real microphone or a capture card, but is instead sourced by other applications. In our setup we use `aplay` and `ffmpeg` to replay audio/video files into these virtual devices. The in-kernel device emulation is completely transparent to the clients, thus no client-side modification is required.

Another aspect of client emulation is client UI navigation. Each videoconferencing client has client-specific UI elements for interacting with a videoconferencing service, such as logging in, joining/leaving a meeting, switching layouts, etc. We automate UI navigation of deployed clients by emulating various input events (e.g., keyboard typing, mouse activity, screen touch) with OS-specific tools (e.g., `xdotool` for Linux, and `adb-shell` for Android). For each videoconferencing system, we script the entire workflow of its client.

Coordinated client deployments. Fully-emulated clients allow us to deploy the clients in public clouds and/or mobile testbeds for automated testing. The fact that we control audiovisual data feed for the clients as well as their UI navigation provides unique opportunities for us to gain, otherwise difficult to obtain, insights into the videoconferencing systems under test. For example, one client can be injected with a video feed with specific patterns (e.g., periodic ON/OFF signals, or high-/low-motion videos), and other clients receive the feed through a videoconferencing service. By comparing the injected feed and received feeds, we can evaluate different videoconferencing services. We can easily coordinate the activities of multiple participants in a given conferencing session to facilitate our analysis (e.g., only one user's screen is active at a time).

Platform-agnostic data collection. Even with client emulation and coordination, the closed nature of the existing videoconferencing systems (e.g., proprietary client software and end-to-end encryption) poses as a hurdle to comparing the systems with objective and unified metrics. That has led us to perform data collection in a *platform-agnostic* fashion as follows.

First, we derive some of the evaluation metrics from network-level monitoring and measurements. For example, we measure streaming lag by correlating packet timestamps on sender-side and on receiver-side. That way, we can evaluate the videoconferencing infrastructures without being influenced by specifics in client deployments. This, however, requires accurate clock synchronization among deployed clients. Fortunately, major public clouds already provide dedicated time sync services for tenant workloads with their own stratum-1 clock sources [18, 26].

In order to supplement network-based metrics with user-perceived quality metrics, we record videoconferencing sessions from individual participants' perspective, and assess the quality of recorded audios/videos across different platforms. While Zoom provides a local recording option for each participant, other services like Webex or Meet only allow a meeting host to record a session. In the end, we adopt a desktop recording approach as a platform-agnostic measure. We run a videoconferencing client in full screen mode, and use `simplescreenrecorder` to record the desktop screen with audio, within a cloud VM itself.

Finally, we also evaluate the videoconferencing systems from their clients' resource-utilization perspectives, which is particularly important for mobile devices. While these metrics can be influenced by client implementation, we believe that platform-driven factors (e.g., audio/video codecs) may play a bigger role.

3.2 Deployment Targets

Based on the design approach described above, we deploy emulated videoconferencing clients on a group of cloud VMs and Android mobile phones. Each of the cloud VMs hosts a videoconferencing client in a fully emulated setting to generate (via emulated devices) and/or receive a streaming feed, while Android devices only receive feeds from videoconferencing systems without device emulation.[1] In the following, we describe each of these deployment targets in more details.

Cloud VM. A cloud VM runs a videoconferencing client on a remote desktop in a fully-emulated environment. It consists of several components as shown in Fig. 1. *Media feeder* replays audio and video files into corresponding loopback devices. The audio and video files are either synthetically created, or extracted individually from a video clip with sound. *Client monitor* captures incoming/outgoing videoconferencing traffic with `tcpdump`, and dumps the trace to a file for offline analysis, as well as processes it on-the-fly in a separate "active probing" pipeline. In this pipeline, it discovers streaming service endpoints (IP address, TCP/UDP port) from packet streams, and performs round-trip-time (RTT) measurements against them. We use `tcpping` for RTT measurements because ICMP pings are blocked at the existing videoconferencing infrastructures. *Client controller* replays a platform-specific script for operating/navigating a client, including launch, login, meeting-join/-leave and layout change.

In order to host the cloud VMs, a public cloud must meet the following requirements. First, the cloud must *not* be used to operate the videoconferencing systems under test. For example, the majority of Zoom infrastructure is known to be hosted at AWS cloud [1]. If we run our emulated clients in the same AWS environment, Zoom will be heavily favored in our evaluation due to potential intra-cloud network optimization. To prevent such bias, we exclude any public cloud being used by the videoconferencing systems we tested. The cloud must also have reasonably wide geographic coverage. In the end we choose Azure cloud [16] as our benchmarking platform.

Android devices. We use Samsung Galaxy S10 and J3 phones, representative of both *high-end* and *low-end* devices (Table 2). The battery of the J3 is connected to a Monsoon power meter [31] which produces fine-grained battery readings. Both devices are connected to a Raspberry Pi (via WiFi to avoid USB noise on the power readings) which is used to automate Android UI navigation and to monitor their resource utilization (e.g., CPU usage). Both tasks are realized via Android Debugging Bridge (`adb`). The phones connect to the Internet over a fast WiFi – with a symmetric upload and download bandwidth of 50 Mbps. Each device connects to its own WiFi realized by the Raspberry Pi, so that traffic can be easily isolated and captured for each device.

[1]While Android devices can generate sensory data from their onboard camera/microphone, we mostly do not use them for reproducible benchmarking.

Name	Android Ver.	CPU Info	Memory	Screen Resolution
Galaxy J3	8	Quad-core	2GB	720x1280
Galaxy S10	11	Octa-core	8GB	1440x3040

Table 2: Android devices characteristics.

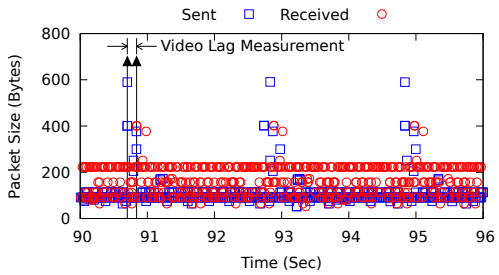


Figure 2: Video lag measurement.

4 QUALITY OF USER EXPERIENCE

In this section, we present QoE analysis results from our benchmark analysis of three major videoconferencing systems: Zoom, Webex and Meet. The experiments were conducted from 4/2021 to 5/2021.

4.1 Cloud VM Setup

Each cloud VM we deploy has 8 vCPUs (Intel Xeon Platinum 8272CL with 2.60GHz), 16GB memory and 30GB SSD. We make sure that the provisioned VM resources are sufficient for all our benchmark tests, which involve device emulation, videoconferencing, traffic monitoring and desktop recording. The screen resolution of the VM's remote desktop is set to 1900×1200. We use the native Linux client for Zoom (v5.4.9 (57862.0110)), and the web client for Webex and Meet since they do not provide a native Linux client.

4.2 Streaming Lag

First we evaluate streaming lag experienced by users (i.e., time delay between audio/video signals ingested by one user and those received by another). While measuring streaming lag in real-life videoconferencing is difficult, our emulated clients with synchronized clocks allow us to quantify the lags precisely. We purposefully set the video screen of a meeting host to be a blank-screen with periodic flashes of an image (with two-second periodicity), and let other users join the session with no audio/video of their own. Using such a bursty, one-way video feed allows us to easily determine the timing of sent/received video signals from network traffic monitoring.

For example, Fig. 2 visualizes the packet streams observed on the meeting host (sender) and another user (receiver). The squares represent the sizes of packets sent by a meeting host, and the circles show the sizes of packets received by a user. As expected there are periodic spikes of "big" packets (>200 bytes) that match periodic video signals sent and received. The first big packet that appears after more than a second-long quiescent period indicates the arrival of a non-blank video signal. We measure streaming lag between the meeting host and the other participant with the time shift between the first big packet on sender-side and receiver-side.

Region	Location	Name	Count
US	Iowa	US-Central	1
	Illinois	US-NCentral	1
	Texas	US-SCentral	1
	Virginia	US-East	2
	California	US-West	2
Europe	Switzerland	CH	1
	Denmark	DE	1
	Ireland	IE	1
	Netherlands	NL	1
	France	FR	1
	London, UK	UK-South	1
	Cardiff, UK	UK-West	1

Table 3: VM locations/counts for streaming lag testing.

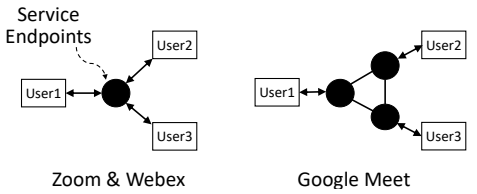


Figure 3: Videoconferencing service endpoints.

Admittedly, this network-based metric discounts any potential delay caused by a receiver-side client (e.g., due to stream buffering/decoding). However, it is effective to evaluate and compare lags induced by streaming infrastructures and their geographic coverage.

In the first set of experiments we deploy seven VMs in the US, as indicated by the "VM count" field in Table 3. We create a meeting session with one VM (in either US-east or US-west) designated as a meeting host, which then broadcasts periodic video signals to the other six participating VMs for two minutes before terminating the session. We collect 35-40 lag measurements from each participant during the session. For more representative sampling, we create 20 such meeting sessions with the same meeting host. Thus in the end we have a total of 700-800 lag measurements from each of the six VMs for a particular meeting host. We repeat this experiment on Zoom, Webex and Meet. In the second set of experiments we use seven VMs deployed in Europe, as shown in Table 3, and redo the above experiments with meeting hosts in UK-west and Switzerland.

We observe that the multi-user sessions created in this experiment are all relayed via platform-operated service endpoints with a designated fixed port number (`UDP/8801` for Zoom, `UDP/9000` for Webex, and `UDP/19305` for Meet).[2] Fig. 3 compares the three videoconferencing systems in terms of how their clients interact with the service endpoints for content streaming, which we discover from their traffic traces. On Zoom and Webex, a single service endpoint is designated for each meeting session, and all meeting participants send or receive streaming data via this endpoint. On Meet, each client connects to a separate (geographically close-by) endpoint, and meeting sessions are relayed among these multiple distinct endpoints.

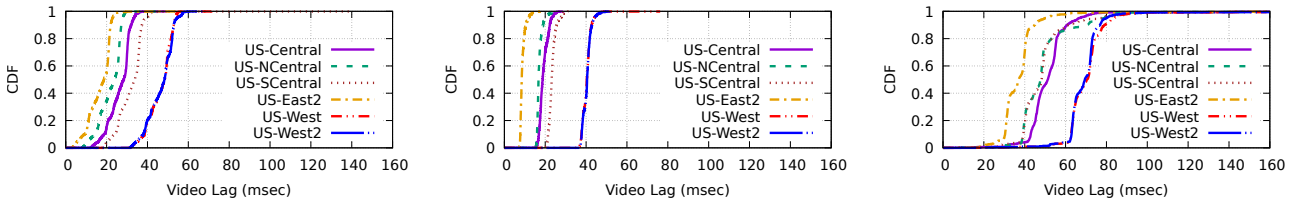
The number of distinct service endpoints encountered by a client varies greatly across different platforms. For example, out of 20 videoconferencing sessions, a client on Zoom, Webex and Meet encounters, on average, 20, 19.5 and 1.8 endpoints, respectively. On Zoom and Webex, service endpoints almost always change (with different IP addresses) across different sessions, while, on Meet, a client tends to stick with one or two endpoints across sessions.

Figs. 4–7 plot the CDFs of streaming lag experienced by clients in four different scenarios. In Figs. 4 and 5, we consider videoconferencing sessions among seven US-based clients (including a meeting host), where the host is located in either US-east or US-west. In Figs. 6 and 7, similarly we set up videoconferencing sessions among seven clients in Europe, with a meeting host in either UK or Switzerland. We make the following observations from the results.

4.2.1 US-based Videoconferencing. When sessions are created from US-east (Fig. 4), across all three platforms, streaming lag experienced by clients increases as they are further away from US-east, with the US-west clients experiencing the most lags (about 30 ms higher than the US-east client). This implies that streaming is relayed via the servers in US-east, where the meeting host resides. This is in fact confirmed by Fig. 8, where we plot RTTs between clients and service endpoints they are connected to. In the figure, RTTs measured by different clients are indicated with distinct dots. Each dot represents an average RTT (over 100 measurements) in a particular session. On Zoom and Webex, RTTs measured by US-east users are much lower than those by US-west users. On Meet, RTTs are uniform across clients due to its distributed service endpoint architecture as shown in Fig. 3.

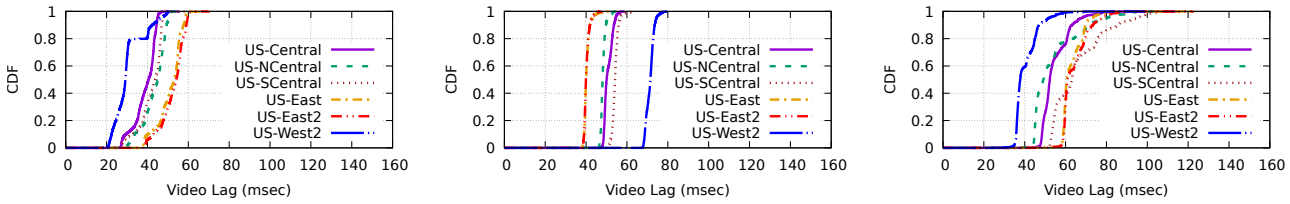
When a meeting host is in US-west (Fig. 5), geographic locality plays a similar role with Zoom and Meet, where the most far-away clients in US-east experience the worst lags. In case of Webex, however, the worst streaming lag is actually experienced by another user in US-west. According to the RTTs collected in this scenario for Webex (Fig. 9b), its service endpoints seem to be provisioned on the *east*-side of the US even when sessions are created in US-west, causing the streaming between US-west users to be detoured via US-east. Due to the geographically-skewed infrastructure, the lag distributions for US-west-based sessions are simply shifted by 30 ms from the US-east-based counterparts (Figs. 4b and 5b). One unexpected observation is that Meet sessions exhibit the worst lag despite having the lowest RTTs. This might be due to the fact that Meet sessions are relayed via *multiple* service endpoints, unlike Zoom and Webex (Fig. 3). In addition, although Meet's videoconferencing infrastructure appears to be distributed over wider locations (with lower RTTs), the total aggregate server capacity at each location may be smaller, hence leading to more load variation.

[2] One exceptional case we observe is that, on Zoom, if there are only two users in a session, peer-to-peer streaming is activated, where they stream to each other *directly* on an ephemeral port without going through an intermediary service endpoint.



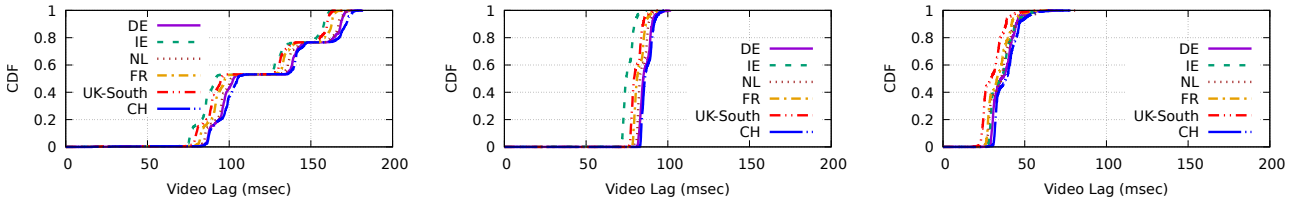
(a) Zoom (b) Webex (c) Meet

Figure 4: CDF of streaming lag: meeting host in US-east.



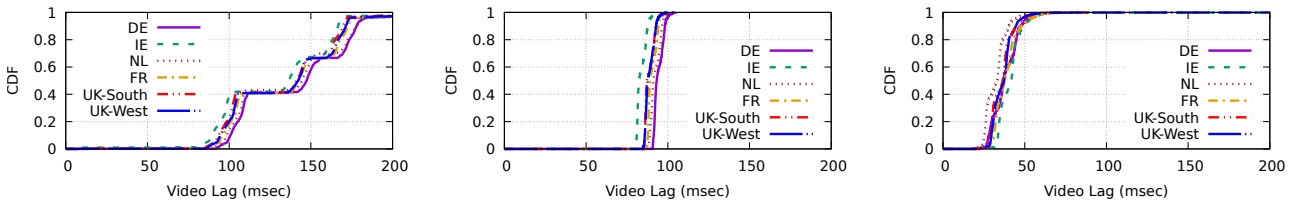
(a) Zoom (b) Webex (c) Meet

Figure 5: CDF of streaming lag: meeting host in US-west.



(a) Zoom (b) Webex (c) Meet

Figure 6: CDF of streaming lag: meeting host in UK-west.



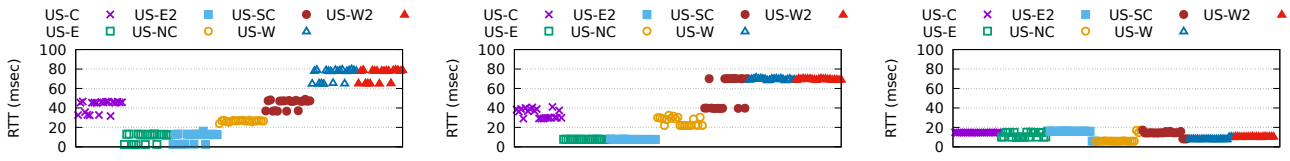
(a) Zoom (b) Webex (c) Meet

Figure 7: CDF of streaming lag: meeting host in Switzerland.

4.2.2 Non-US-based videoconferencing. According to Figs. 6–7, when sessions are set up among clients in Europe, Zoom/Webex clients experience much higher lags than Meet users. When compared to the Zoom/Webex sessions created in US-east, clients in Europe experience 55–75 ms and 45–65 ms higher median lags on Zoom and Webex, respectively. The reported RTTs (Figs. 10 and 11) show that the clients closer to the east-coast of US (e.g., UK and Ireland) have lower RTTs than those located further into central Europe (e.g., Germany and Switzerland). These observations suggest that the service infrastructures used are located somewhere in US.[3]

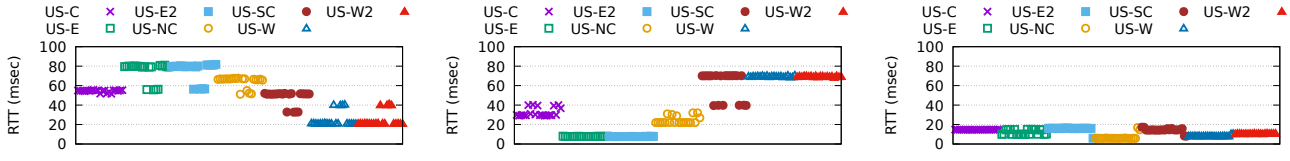
Comparing Zoom and Webex, one can see that RTTs to service endpoints on Zoom vary much more widely across different sessions. In fact, as shown in Figs. 10a and 11a, RTTs tend to spread across three distinct ranges that are 20 ms and 40 ms apart, which causes step-wise lag distributions in Figs. 6a and 7a. This suggests that Zoom may be employing *regional load balancing* within the US when serving non-US sessions. Whereas in Webex, RTTs to service endpoints consistently remain close to the trans-Atlantic RTTs [15], indicating that non-US sessions are relayed via its infrastructure in US-east. In case of Meet, its distributed service endpoints allow clients in Europe to enjoy stream lags that are comparable to US-based counterparts without any artificial detour. The reason why its streaming lag is lower

[3]We are not able to pinpoint the locations of the infrastructures because `traceroute`/`tcptraceroute`-probings are all blocked.



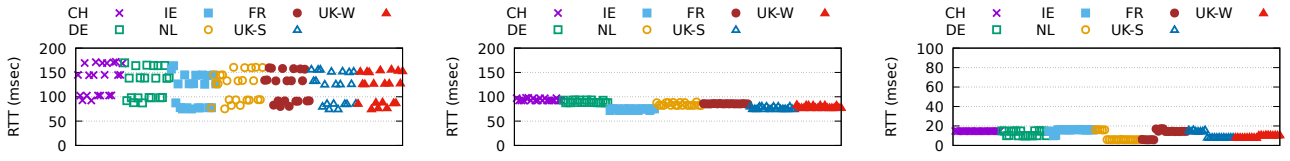
(a) Zoom (b) Webex (c) Meet

Figure 8: Service proximity: meeting host in US-east.



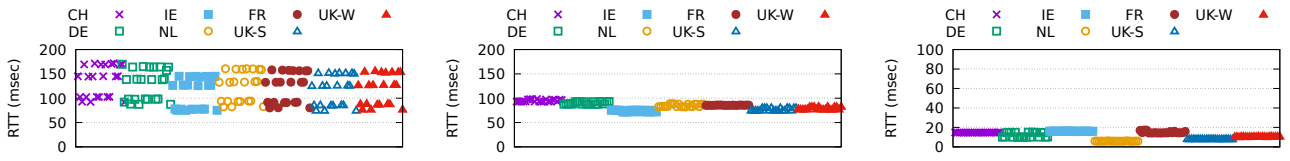
(a) Zoom (b) Webex (c) Meet

Figure 9: Service proximity: meeting host in US-west.



(a) Zoom (b) Webex (c) Meet

Figure 10: Service proximity: meeting host in UK-west.



(a) Zoom (b) Webex (c) Meet

Figure 11: Service proximity: meeting host in Switzerland.

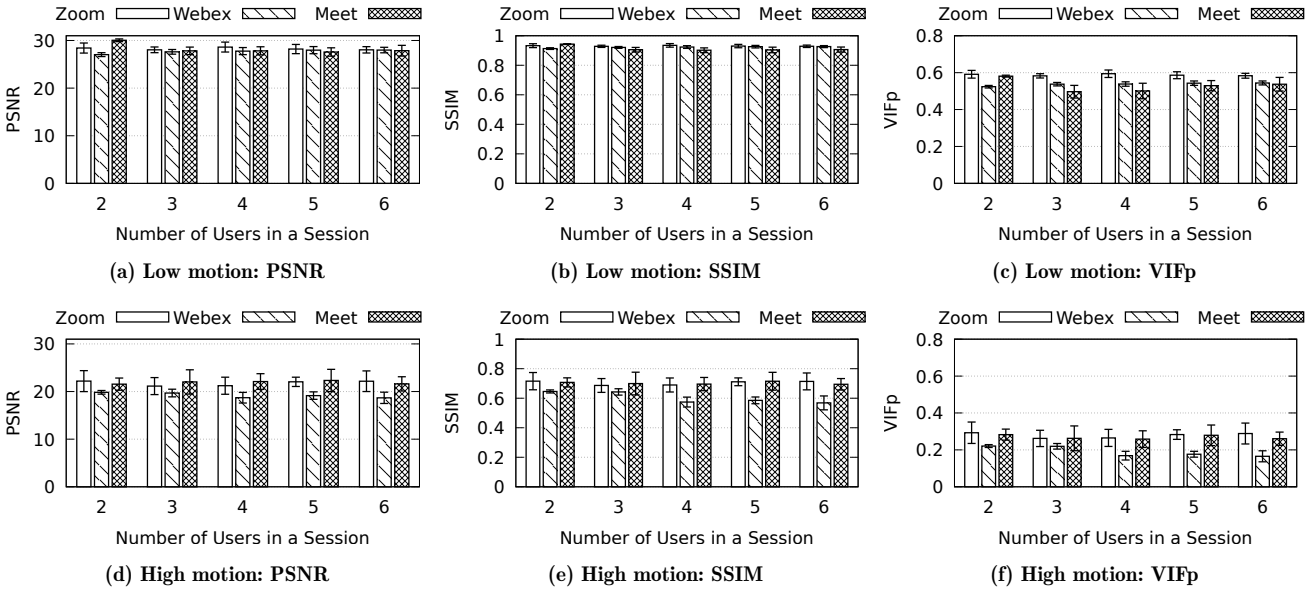
in Europe than in the US may be because the end-to-end latency among the clients (connected via service endpoints) in Europe may be smaller than that in the US. Average RTT within Europe is indeed smaller than that in the US [15].

4.3 User-Perceived Video Quality

Next we shift our focus to user-perceived quality of videoconferencing. A videoconferencing client typically captures a single person view against a stationary background, but it is also possible to have high-motion features in streamed content if a participant is joining a session from a mobile device, sharing a video playback with dynamic scenes, or showing a media-rich presentation, etc. In general, however, little is known about how different videoconferencing systems measure up to one another in terms of user-perceived quality under different conditions.

For this evaluation we prepare two distinct video feeds with 640×480 resolution: (i) a low-motion feed capturing the upper body of a single person talking with occasional hand gestures in an indoor environment, and (ii) a high-motion tour guide feed with dynamically moving objects and scene changes. On each videoconferencing system, we use a designated meeting host VM to create 10 five-minute long sessions, and inject the low-/high-motion videos into the sessions in an alternating fashion (hence two sets of five sessions). In each session, we let N clients join the session and render the received video feed in *full screen* mode while their desktop screen is recorded locally. For desktop recording, we use `PulseAudio` as audio backend, and set the video/audio codec to `H.264` (30 fps) and `AAC` (128 Kbps), respectively. We repeat the whole experiment as we vary N from one to five.[4]

[4] A typical number of users in a videoconferencing session is less than five [22].



(a) Low motion: PSNR (b) Low motion: SSIM (c) Low motion: VIFp

(d) High motion: PSNR (e) High motion: SSIM (f) High motion: VIFp

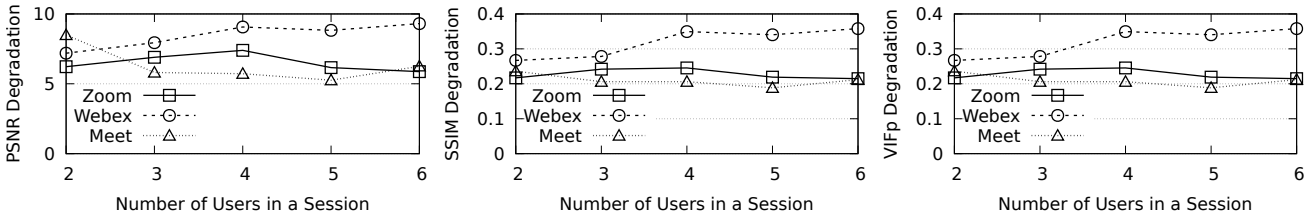
Figure 12: Video QoE metrics comparison (US).

Figure 13: Video screen with padding.

We compare the originally injected videos and recorded videos in terms of their quality with `VQMT` [6]. The `VQMT` tool computes a range of well-known objective QoE metrics, including PSNR (Peak Signal-to-Noise Ratio), SSIM (Structural Similarity Index Measure) [39] and VIFp (Pixel Visual Information Fidelity) [35]. Each of these metrics produces *frame-by-frame similarity* between injected/recorded videos. We take an average over all frames as a QoE value. One issue that complicates accurate quality comparison is the fact that the video screen rendered by a client is partially blocked by client-specific UI widgets (e.g., buttons, user thumbnails, etc.), even in full screen mode. To avoid such partial occlusion inside the video viewing area, we prepare video feeds with enough padding (Fig. 13). When recorded videos are obtained, we perform the following post-processing on them before analysis. We first crop out the surrounding padding and resize video frames to match the content layout and resolution of the injected videos. On top of that, we synchronize the start/end time of original/recorded videos with millisecond-level precision by trimming them in a way that per-frame SSIM similarity is maximized.

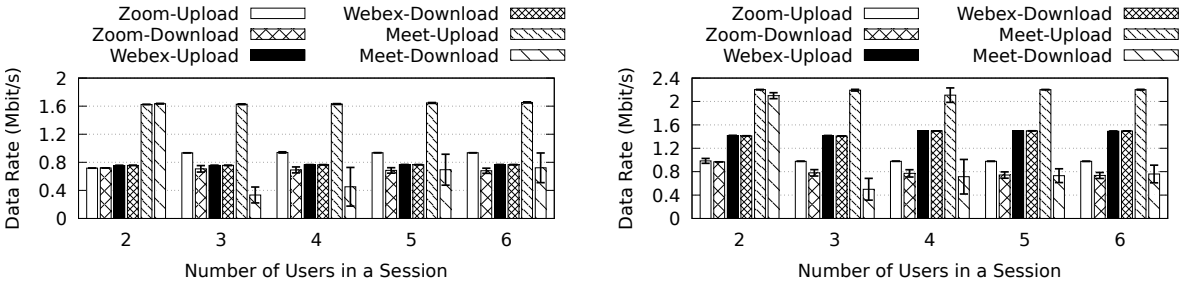
4.3.1 US-based Videoconferencing. We use one cloud VM in US-east designated as a meeting host which broadcasts a stream, and up to five other VMs in US-west and US-east receiving the stream as passive participants. Fig. 12 compares the quality of video streaming for these VMs in terms of three QoE metrics (PSNR, SSIM & VIFp) as the number of users in a session (N) increases. The height of bars indicates average QoE values across all sessions, with the errorbars being standard deviations. For easy comparison between low-motion and high-motion feeds, Fig. 14 shows the amount of QoE reduction with high-motion feeds (compared to low-motion feeds). Figs. 15a and 15b show the corresponding data rates for these sessions.

We make the following observations from the figures. Comparing Figs. 12a–12c against Figs. 12d–12f, one can find that, across all three platforms, low-motion sessions experience less quality degradation than high-motion sessions because their video feeds contain largely static background. The amount of decrease in QoE values between low-motion/high-motion sessions (Fig. 14) is significant enough to downgrade mean opinion score (MOS) ratings by one level [30]. On Webex, QoE degradation in high-motion scenario tends to become more severe with more users. Whereas no such consistent pattern is observed in Zoom and Meet. The QoE results from low-motion sessions (Figs. 12a–12c) show that there is a non-negligible QoE drop between N=2 and N>2 on Meet. We find that, on Meet, the data rate for two-user sessions (1.6–2.0 Mbps) is significantly higher than other multi-user sessions (0.4–0.6 Mbps) (Fig. 15). Such higher traffic rate with N=2 helps with the QoE of low-motion sessions, but does not contribute much to the QoE of high-motion sessions. Among the three, Webex exhibits the most *stable* QoE across



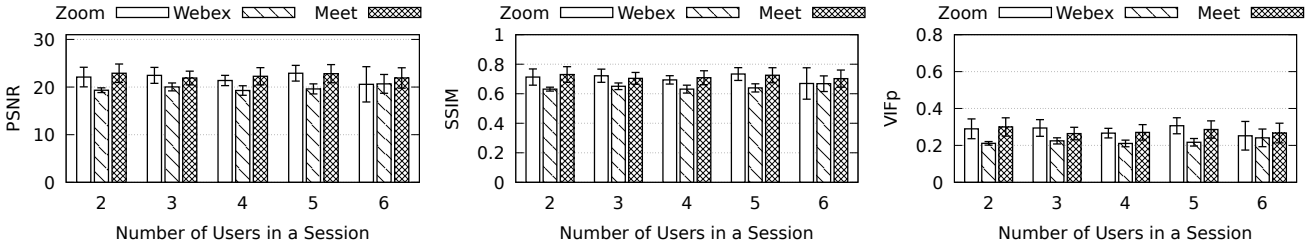
(a) PSNR (b) SSIM (c) VIFp

Figure 14: Video QoE reduction when video feeds are changed from low-motion to high-motion (US).



(a) US: low-motion (b) US: high-motion

Figure 15: Upload/download data rates (US). The data rate is computed from Layer-7 payload length in pcap traces. "X-Upload" shows the average upload rate of a meeting host (using Zoom, Webex, or Meet) who is broadcasting a video feed, while "X-Download" indicates the average download rate of the clients who are receiving the feed. All uploads/downloads occur via cloud-based relay, except for Zoom with N=2 which uses peer-to-peer traffic.



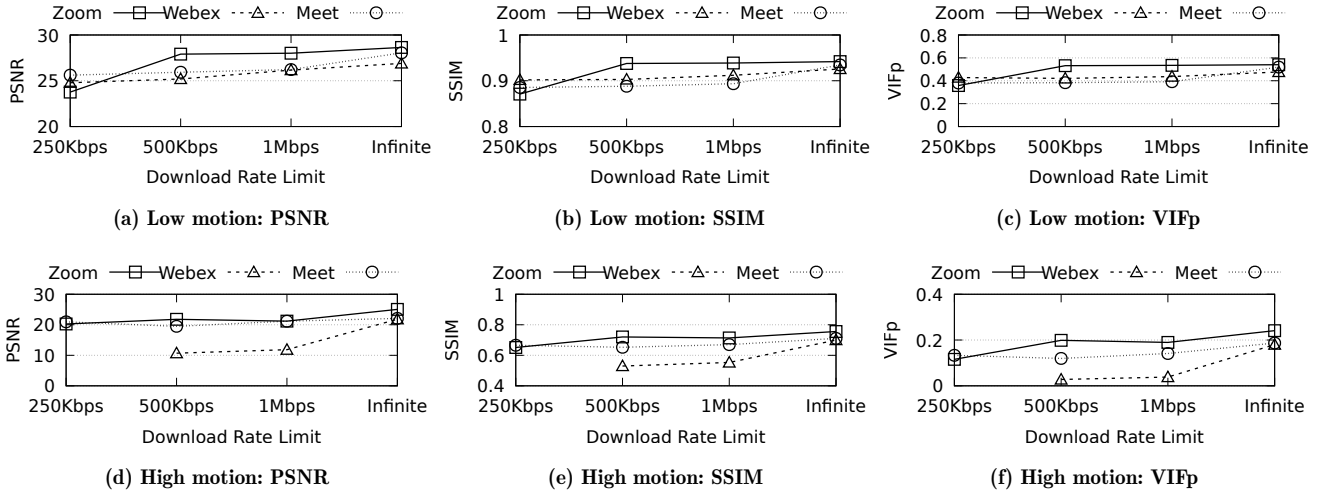
(a) High motion: PSNR (b) High motion: SSIM (c) High motion: VIFp

Figure 16: Video QoE metrics comparison (Europe).

different sessions. A similar behavior is observed in Figs. 4–7, where Webex shows the least variance in streaming lag as well.

Traffic-wise (Fig. 15), we focus on two aspects: (1) data rate difference between low-motion and high-motion feeds, and (2) data rate variation across multiple sessions with the same feed. All three systems send out a low-motion video feed in a lower rate than a high-motion counterpart as the former is more compressible. The rate reduction in low-motion streams is the highest in Webex, where its low-motion sessions almost halve the required downstream bandwidth. With a given video feed, Webex shows virtually no fluctuation in data rate across multiple sessions. On the other hand, Meet reduces its data rate in low-motion sessions roughly by 20% compared to high-motion sessions, but with much more *dynamic* rate fluctuation across different sessions than Zoom/Webex. Zoom exhibits the least difference (5–10%) in data rate between low-motion and high-motion sessions. Its downstream data rate is slightly higher with peer-to-peer streaming (∼1 Mbps; N=2) than with cloud-based relay (∼0.7 Mbps; N>2). When QoE metrics and data rates are considered together, Zoom appears to deliver the best QoE in the most bandwidth-efficient fashion, at least in the US.

4.3.2 Non-US-based videoconferencing. We repeat the QoE analysis experiment using a set of VMs created in Europe, one VM in Switzerland designated as a meeting host, and up to five other VMs (in France, Germany, Ireland, UK) joining the session created by the host. Fig. 16 shows QoE values on three systems with high-motion sessions. The results from



(a) Low motion: PSNR (b) Low motion: SSIM (c) Low motion: VIFp

(d) High motion: PSNR (e) High motion: SSIM (f) High motion: VIFp

Figure 17: Effect of bandwidth constraints on video quality.

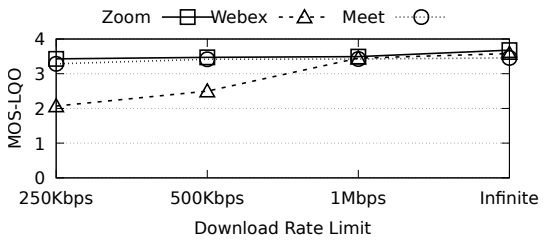


Figure 18: Audio quality under bandwidth constraints. MOS-LQO is computed in speech mode for low-motion sessions which contain only human voice.

low-motion sessions are similar to those collected in the US, and thus are omitted. When compared side-by-side, Meet maintains a slight edge in QoE metrics among three systems, potentially due to its European presence. In case of Zoom, although its average QoE appears to be similar to that of Meet, its QoE variation across different sessions is higher than Meet for high N. This observation is aligned with our earlier finding in Fig. 6a, where we show that its regional load balancing causes more variable streaming lag in Europe.

4.4 Streaming under Bandwidth Constraints

The experiments presented so far are conducted in an *unlimited* bandwidth environment. The cloud VMs used have a bidirectional bandwidth of multi-Gbps [3], which far exceeds the measured data rates of 1–2 Mbps (Fig. 15). In the next set of experiments, we apply artificial bandwidth caps on our cloud VM and examine its effect on QoE. We use Linux `tc/ifb` modules to enable traffic shaping on incoming traffic. Here we present QoE metric analysis not just for video but also for audio. We extract video and audio data separately from recorded videoconferencing sessions. For audio QoE analysis, we perform the following processing on extracted audio. First, we normalize audio volume in the recorded audio (with EBU R128 loudness normalization), and then synchronize the beginning/ending of the audio in reference to the originally injected audio. We use the `audio-offset-finder` tool for this. Finally, we use the `ViSQOL` tool [19] with the original/recorded audio data to compute the MOS-LQO (Mean Opinion Score - Listening Quality Objective) score, which ranges from 1 (worst) to 5 (best).

Figs. 17 and 18 show how video/audio QoE metrics change under various rate-limiting conditions. Each dot in the figures represents the average QoE values of five 5-minute long sessions.

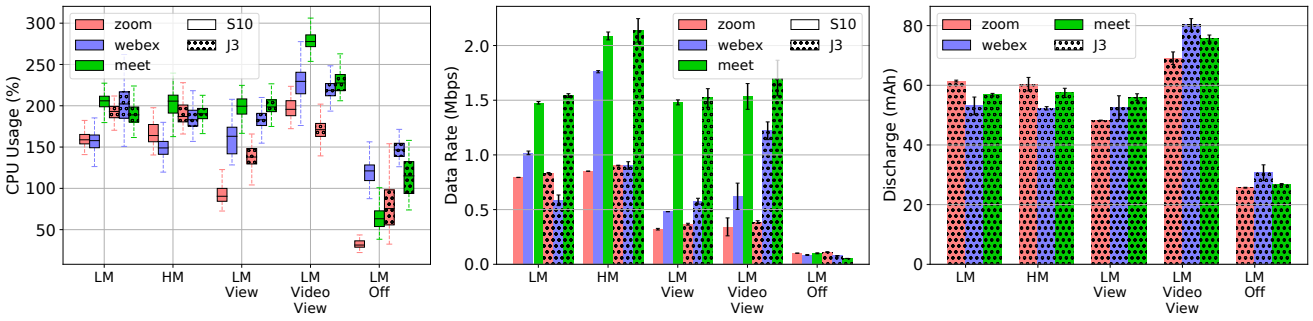
Video QoE. Overall, Zoom tends to maintain the best QoE with decreasing bandwidth limits, although there is sudden drop in QoE with a bandwidth cap of 250 Kbps. Meet maintains more graceful QoE degradation across all scenarios. Webex suffers from the most significant QoE drops with smaller bandwidth caps. With bandwidth $\leq$ 1 Mbps, video frequently stalls and even completely disappears and reappears on Webex.

Audio QoE. Compared to non-negligible QoE drops in video, audio QoE levels remain virtually constant on Zoom and Meet, probably due to the low data rate of audio (90 Kbps for Zoom and 40 Kbps for Meet).[5] However, voice quality on Webex, even with its low rate (45 Kbps), is relatively sensitive to bandwidth limits, starting to deteriorate noticeably (e.g., manifested as distorted/paused sound) with a limit of 500 Kbps or less.

5 RESOURCE CONSUMPTION

After having investigated user-perceived QoE offered by three major videoconferencing systems, we shift our attention to their client-side *resource consumption*, such as CPU, bandwidth and battery usages. For this analysis, we resort to

[5]We measure their audio rates separately using audio-only streams.



(a) CPU usage (b) Data rate (c) Battery discharge (J3 only)

Figure 19: Resource consumption evaluation. Samsung S10 and J3 (Android).

two Android devices: S10 (high-end) and J3 (low-end) as described in Table 2. In addition to these devices, each experiment includes one cloud VM designated as a meeting host. Since the Android devices are located in a residential access network of the east-coast of US, we run the meeting host in a US-east VM. We set the videoconference duration to five minutes, and repeat each experiment five times. The meeting host streams the two previously introduced video feeds: low-motion (`LM`) and high-motion (`HM`).

At the Android devices, we consider several device/UI settings which can affect videoconferencing sessions. Unless otherwise noted by the label "`Video`", each device's camera is turned off to minimize noise, and the incoming video feed is displayed in full screen. Given the low-motion video, we: 1) change the videoconferencing client's view into a gallery-view (`LM-View`), which assigns to each videoconference participant an equal portion of the screen,[6] 2) turn on the cameras and the gallery-view (`LM-Video-View`), and 3) turn off both camera and screen (`LM-Off`), simulating a driving scenario.

CPU usage. Fig. 19a shows boxplots of CPU usage sampled every three seconds for the experiment duration across all devices and scenarios. Each boxplot accounts for CPU samples collected across five repetitions of an experiment. We report CPU usage in absolute numbers, e.g., 200% implies full utilization of two cores. If we focus on the `LM` and `HM` scenarios for S10 (high-end), the figure shows that Zoom and Webex have comparable CPU usage (median of 150–175%), while Meet adds an extra 50%. When we focus on J3 (low-end device), CPU usage among the three clients is instead comparable (median around 200%). This indicates a dynamic behavior of the Meet client which only grabs more resources if available.

Zoom is the only client which benefits from the gallery view (both when the device's camera is on or off), reducing its CPU usage by 50% on both devices. Meet sees no benefit from this setting, which is expected given that it has no direct support for it, *i.e.,* the meeting host's video still occupies about 80% of the screen. Surprisingly, Webex does not benefit from its gallery view, even causing a slight CPU increase on S10. Irrespective of the videoconferencing client, activating the device's camera (`LM-Video-View`) adds an extra 100% and 50% of CPU usage on S10 and J3, respectively. The higher CPU usage on S10 is due to a better camera with twice as many megapixels (10M), HDR support, etc.

Finally, CPU usage is minimized when the device screen is off. However, while Zoom and Meet's CPU usage is reduced to 25–50% (S10), Webex still requires about 125%. This result, coupled with the lack of benefit of Webex's gallery setting, indicates some room for Webex to improve their Android client with more careful detection of user settings, as achieved by its competitors.

Data rate. We now focus on the Layer-7 download data rate, computed directly from pcap traces. Fig. 19b shows the average download data rate per client, device, and setting, with errorbars reporting the standard deviation. If we focus on the high-end device (S10) and the `LM`/`HM` scenarios, the figure shows a trend similar to Fig. 19a: Zoom uses the lowest data rate while Meet the highest, and the HM video causes a significant data rate increase, with the exception of Zoom. When focusing on the low-end device (J3), we notice that only Webex shows a "truly" adaptive behavior, *i.e.,* lower data rate for both LM and the low-end device. Zoom instead sticks to a somehow default data rate (750 Kbps), while Meet only takes into account the quality of the video, and not the target device. As previously observed for CPU usage, Meet's data rate is not impacted by the client-side gallery view, while it drops Zoom's data rate by 50%, both with a device's video on and off. Webex's gallery view is instead less data efficient, particularly when a device's video is on. In this case, J3 reports a significant increase in the data rate (more than doubled compared to the `LM` scenario) due to the video sent by S10. In comparison, S10 reports a much lower video rate (700 Kbps vs. 1.2 Mbps) due to the J3's lower quality camera, as well as lack of light due to its location in the lab. Finally, the `LM-Off` scenarios confirm that no video is streamed when the screen is off, and just 100–200 Kbps are needed for audio.

[6] Meet has no support for this feature. We approximate it by "zooming out", *i.e.,* revealing UI controls like mute or hang up.

Table 4: Data rate and CPU usage with various videoconference sizes (N). Each cell reports statistics for S10/J3.

N	Client	Full screen		Gallery	
		Data rate (Mbps)	CPU (%)	Data rate (Mbps)	CPU (%)
3	Zoom	0.85/0.9	164/186	0.33/0.37	102/148
	Webex	1.76/0.9	148/183	0.57/0.59	149/186
	Meet	2.08/2.13	205/190	2.08/2.11	209/200
6	Zoom	0.92/0.94	189/212	0.71/0.73	101/152
	Webex	1.75/0.9	140/195	0.43/0.47	155/184
	Meet	2.25/2.33	257/211	2.15/2.24	235/219
11	Zoom	0.91/0.96	191/211	0.73/0.75	100/150
	Webex	1.76/0.89	141/194	0.48/0.43	154/182
	Meet	2.24/2.36	258/210	2.16/2.26	236/220

Battery usage. Next, we report on the battery consumption associated with videoconferencing. In this case, we only focus on J3, whose (removable) battery is connected to a Monsoon power meter [31]. Fig. 19c confirms that videoconferencing is an expensive task on mobile, draining up to 40% of its 2600mAh battery during an one-hour conference with camera on. Overall, the figure shows no dramatic difference among the three clients, whose battery usage is within 10% of each other. Zoom is the most energy efficient client with gallery view (`LM-View`), which provides a 20% reduction compared with `LM`. Gallery view does not provide benefits to both Webex and Meet, similar to what we reported for both CPU usage and data rate. Finally, turning off the screen and relying on audio only saves up to 50% of a user's battery.

Videoconference size. Finally, we investigate the impact of the number of conference participants on client-side resource utilization. To do so, in addition to the meeting host, we introduce up to eight cloud-VMs as participants. To further stress the devices under test, we configure the host as well as the extra eight cloud VMs to stream a high-motion video simultaneously. We consider two UI settings in the Android clients: *full screen* and *gallery*.

Table 4 summarizes the results; we report on average data rate and median CPU utilization per device (S10/J3, in each cell) and scenario. Note that $N = 3$ is the scenario we have evaluated so far, *i.e.,* one cloud VM as a meeting host plus two Android devices. In full screen mode, Meet incurs a significant increase in both data rate (10%) and CPU usage (50%). This is because, even in full screen, Meet still shows a small preview of the video of the other two participants (plus the device's video, if on). Conversely, Zoom and Webex appear visually equivalent in full screen regardless of N. Still, for Zoom we observe a small increases in its data rate (5%) and CPU (12%) suggesting that some additional video streams are being buffered in the background with the goal to minimize latency in case a user decides to change the view into *gallery*. Although we could not capture such latency, we visually confirmed that both Zoom and Meet allow us to rapidly switch between participants, while Webex incurs some buffering time in the order of seconds.

If we focus on the gallery view, we see that the extra participants cause a twofold data rate increase for Zoom, but no additional CPU usage. A similar trend is observed for Webex CPU-wise, but in this case we also observe a counter-intuitive data rate reduction (from 600 Kbps down to 450 Kbps). Upon visual inspection, we find that this reduction is associated with a significant quality degradation in the video stream. Further increasing the participants to 11 does not lead to additional resource consumption. This happens because the gallery view of Zoom and Webex – as well as the only Meet's setting – show videos for up to four concurrent participants.

6 LIMITATIONS

We conclude the paper by discussing several limitations of our study and future works we plan to explore.

Effect of last mile. Our cloud-based vantage points may not represent the realistic last-mile network environments (e.g., broadband, wireless) of typical videoconferencing users from two perspectives. First, the upstream connectivity of cloud-emulated clients (with multi-Gbps available bandwidth) is too idealistic. While our experiments with bandwidth emulation (Section 4.4) and a mobile testbed (Section 5) are intended to address this limitation, a more realistic QoE analysis would consider dynamic bandwidth variation and jitter as well. Another caveat is that all our emulated clients are created inside a *single* provider network (i.e., Azure network). Thus any particular connectivity/peering of the Azure network might influence our cloud experiments. Ideally, the testbed should be deployed across multiple cloud providers to mitigate any artifact of a single provider, or even encompass distributed edge-based platforms provisioned across heterogeneous access networks (e.g., residential [20, 38], campus [2] and enterprise networks [11, 12]). In fact, moving the evaluation platform to the edge would allow us to extend the list of target videoconferencing systems to study.[7]

Free-tier vs. paid subscription. The validity of our findings is limited to the free-tier services. Given the prevalent multi-CDN strategies [9] and differentiated product offerings, user's QoE can change under different subscription plans or any special license arrangements. For example, Zoom is known to allow users with a university license to connect to AWS in Europe [34], which we do not observe with its free-tier/paid subscription plans. In case of Webex, we confirm that, with a paid subscription, its clients in US-west and Europe can stream from geographically close-by Webex servers (with RTTs < 20 ms). Our methodology allows us to easily extend the scope of our study beyond the free-tier services.

Videoconferencing scalability. Our QoE analysis targets small-size videoconferencing sessions (with up to 11 participants). An interesting question is how well user's QoE on each system scales as a videoconferencing session is joined by a moderate/large number of participants. One possible approach would use a mix of crowd-sourced human users (who would generate varied-size videoconferencing sessions) and cloud VMs (which would be under our control for detailed QoE analysis of such sessions).

[7] The use of Azure cloud prevents us from including Microsoft Team in our evaluation list.

Black-box testing. Our measurement study is a case of *black-box testing*, where we do not have any knowledge on inner workings of individual systems we study. As such we are severely limited in our ability to *explain* some of the observations we are making. That said, we still argue that our study presents a valuable contribution to the community. For one, our platform-agnostic methodology is general enough for any arbitrary videoconferencing systems. Also, the proposed evaluation scenarios can be a useful input to videoconferencing operators for enhancing their infrastructures and clients.

Videoconferencing client. Our emulated client runs on Linux only (via in-kernel virtual devices). We consider that the modern Linux environment is representative enough for videoconferencing due to the good Linux support [13] or the use of cross-platform web clients by the existing systems. For completeness, one can extend the client emulation beyond Linux, at least in a desktop environment, using similar device emulation and workflow automation tools on Windows [5, 17] and MacOS [4, 10]. The QoE analysis could then be extended to cover different mobile and desktop environments in a more comprehensive fashion.

REFERENCES

[1] 2020. AWS and Zoom Extend Strategic Relationship. https://press.aboutamazon.com/news-releases/news-release-details/aws-and-zoom-extend-strategic-relationship/.
[2] 2020. CloudLab. https://www.cloudlab.us.
[3] 2020. Fsv2-series – Azure Virtual Machines. https://docs.microsoft.com/en-us/azure/virtual-machines/fsv2-series.
[4] 2020. OBS (macOS) Virtual Camera. https://github.com/johnboiles/obs-mac-virtualcam.
[5] 2020. OBS-VirtualCam. https://github.com/CatxFish/obs-virtual-cam.
[6] 2020. VQMT: Video Quality Measurement Tool. https://www.epfl.ch/labs/mmspg/downloads/vqmt/.
[7] 2021. https://support.zoom.us/hc/en-us/articles/201362023-System-Requirements-for-PC-Mac-and-Linux.
[8] 2021. https://help.webex.com/en-us/WBX22158/What-are-the-Minimum-Bandwidth-Requirements-for-Sending-and-Receiving-Video-in-Cisco-Webex-Meetings.
[9] 2021. 2020 Pandemic Network Performance. Broadband Internet Technical Advisory Group. https://www.bitag.org/documents/bitag_report.pdf.
[10] 2021. Automator User Guide for Mac. https://support.apple.com/guide/automator/.
[11] 2021. AWS Snow Family. https://aws.amazon.com/snow/.
[12] 2021. Azure Stack Edge. https://azure.microsoft.com/en-us/products/azure-stack/edge/.
[13] 2021. Desktop client, mobile app, and web client comparison. https://support.zoom.us/hc/en-us/articles/360027397692-Desktop-client-mobile-app-and-web-client-comparison.
[14] 2021. Google Meet hardware requirements. https://support.google.com/meethardware/answer/4541234.
[15] 2021. IP Latency Statistics by Verizon. https://www.verizon.com/business/terms/latency/.
[16] 2021. Microsoft Azure. https://azure.microsoft.com.
[17] 2021. Power Automate - Microsoft Power Platform. https://powerautomate.microsoft.com.
[18] 2021. Time sync for Linux VMs in Azure. https://docs.microsoft.com/en-us/azure/virtual-machines/linux/time-sync.
[19] 2021. ViSQOL: Perceptual Quality Estimator for speech and audio. https://github.com/google/visqol.
[20] Hyunseok Chang, Adiseshu Hari, Sarit Mukherjee, and T. V. Lakshman. 2014. Bringing the Cloud to the Edge. In *Proc. IEEE Workshop on Mobile Cloud Computing*.
[21] Ana-Paula Correia, Chenxi Liu, and Fan Xu. 2020. Evaluating Videoconferencing Systems for the Quality of the Educational Experience. *Distance Education* 41, 4 (2020), 429–452.
[22] Augusto Espin and Christian Rojas. 2021. The Impact of the COVID-19 Pandemic on the Use of Remote Meeting Technologies. SSRN. https://dx.doi.org/10.2139/ssrn.3766889.
[23] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. 2020. The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic. In *Proc. ACM Internet Measurement Conference (IMC)*.
[24] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *Proc. NSDI*.
[25] Pan Hu, Rakesh Misra, and Sachin Katti. 2019. Dejavu: Enhancing Videoconferencing with Prior Knowledge. In *Proc. the 20th International Workshop on Mobile Computing Systems and Applications*. 63–68.
[26] Randall Hunt. 2017. Keeping Time With Amazon Time Sync Service. https://aws.amazon.com/blogs/aws/keeping-time-with-amazon-time-sync-service/.
[27] Lisa M. Koonin, Brooke Hoots, Clarisse A. Tsang, Zanie Leroy, Kevin Farris, Brandon Jolly, Peter Antall, Bridget McCabe, Cynthia B.R. Zelis, Ian Tong, and Aaron M. Harris. 2020. Trends in the Use of Telehealth During the Emergence of the COVID-19 Pandemic—United States, January–March 2020. *Morbidity and Mortality Weekly Report* 69, 43 (Oct. 2020).
[28] Craig Labovitz. 2020. Network traffic insights in the time of COVID-19: March 23-29 update. https://www.nokia.com/blog/network-traffic-insights-time-covid-19-march-23-29-update/.
[29] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. 2021. Measuring the Performance and Network Utilization of Popular Video Conferencing Applications. *arXiv preprint arXiv:2105.13478* (2021).
[30] Arghir-Nicolae Moldovan and Cristina Hava Muntean. 2017. QoE-aware Video Resolution Thresholds Computation for Adaptive Multimedia. In *Proc. IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*.
[31] Monsoon Solutions Inc. 2021. High Voltage Power Monitor. https://www.msoon.com.
[32] Adam Ozimek. 2020. *Economist Report: Future Workforce*. Technical Report. ClearlyRated. https://www.upwork.com/press/releases/economist-report-future-workforce.
[33] Otto Parra and Maria Fernanda Granda. 2021. Evaluating the Meeting Solutions Used for Virtual Classes in Higher Education during the COVID-19 Pandemic. In *Proc. the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*.
[34] Constantin Sander, Ike Kunze, Klaus Wehrle, and Jan Rüth. 2021. Video Conferencing and Flow-Rate Fairness: A First Look at Zoom and the Impact of Flow-Queuing AQM. In *Proc. International Conference on Passive and Active Measurement*.
[35] Hamid Rahim Sheikh and Alan C. Bovik. 2006. Image Information and Visual Quality. *IEEE Transactions on Image Processing* 15, 2 (2006).
[36] Ravinder Singh and Soumya Awasthi. 2020. Updated Comparative Analysis on Video Conferencing Platforms-Zoom, Google Meet, Microsoft Teams, WebEx Teams and GoToMeetings. *EasyChair Preprint no. 4026* (2020).
[37] Sri Srinivasan. 2020. Cisco Webex: Supporting customers during this unprecedented time. https://blog.webex.com/video-conferencing/cisco-webex-supporting-customers-during-this-unprecedented-time/.
[38] Matteo Varvello, Kleomenis Katevas, Mihai Plesa, Hamed Haddadi, and Benjamin Livshits. 2019. BatteryLab: A Distributed Power Monitoring Platform For Mobile Devices. In *Proc. HotNets '19*.
[39] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 13, 4 (2004).
[40] Cui Zou, Wangchuchu Zhao, and Keng Siau. 2020. COVID-19 Pandemic: A Usability Study on Platforms to Support eLearning. In *Proc. International Conference on Human-Computer Interaction*. Springer, 333–340.

PREDATOR: Proactive Recognition and Elimination of Domain Abuse at Time-Of-Registration

Shuang Hao* Alex Kantchelian† Brad Miller§ Vern Paxson†◇ Nick Feamster‡
*University of California, Santa Barbara †University of California, Berkeley
§Google, Inc. ◇International Computer Science Institute ‡Princeton University
shuanghao@cs.ucsb.edu {akant,vern}@cs.berkeley.edu
bradmiller@google.com feamster@cs.princeton.edu

ABSTRACT

Miscreants register thousands of new domains every day to launch Internet-scale attacks, such as spam, phishing, and drive-by downloads. Quickly and accurately determining a domain's reputation (association with malicious activity) provides a powerful tool for mitigating threats and protecting users. Yet, existing domain reputation systems work by observing domain *use* (*e.g.*, lookup patterns, content hosted)—often too late to prevent miscreants from reaping benefits of the attacks that they launch.

As a complement to these systems, we explore the extent to which features evident at domain registration indicate a domain's subsequent use for malicious activity. We develop PREDATOR, an approach that uses only time-of-registration features to establish domain reputation. We base its design on the intuition that miscreants need to obtain many domains to ensure profitability and attack agility, leading to abnormal registration behaviors (*e.g.*, burst registrations, textually similar names). We evaluate PREDATOR using registration logs of second-level .com and .net domains over five months. PREDATOR achieves a 70% detection rate with a false positive rate of 0.35%, thus making it an effective—and early—first line of defense against the misuse of DNS domains. It predicts malicious domains when they are registered, which is typically days or weeks earlier than existing DNS blacklists.

CCS Concepts

•Security and privacy → Intrusion/anomaly detection and malware mitigation; •Networks → *Network domains;*

Keywords

Domain Registration; Reputation System; Early Detection.

1. INTRODUCTION

The Domain Name System (DNS), the Internet's lookup service for mapping names to IP addresses, provides a critical service for Internet applications. Attackers, however, abuse the DNS service to direct victims to Web sites that host scams, malware, and other malicious content [18, 53]. To mitigate these threats, operators routinely build reputation systems for domain names to indicate whether they are associated with malicious activity. A common mode for developing reputation for DNS domain names is to develop a *blacklist* that curates "bad domains". A network operator who wishes to defend against an attack may use a domain blacklist to help determine whether certain traffic or infrastructure is associated with malicious activity.

Unfortunately, curating a DNS blacklist is difficult because of the high rate of domain registrations and the variety of attacks. For example, every day around 80,000 new domains are registered in the .com zone, with a peak rate of over 1,800 registrations in a single five-minute interval. Establishing domain reputation is thus a continual, ongoing process that must be automated, based on the features of the DNS domains. Existing DNS reputation systems primarily use characteristics of DNS lookups to distinguish legitimate domains from malicious ones [3, 4, 5]. Other systems have derived domain reputation by crawling Web pages to discover malicious content [35, 55]. Unfortunately, these existing systems have a number of limitations. (1) The particular resources and vantage points they employ (*e.g.*, URL crawlers, spam traps) provide limited visibility into various attacks across time, and thus may miss malicious activities of certain domains. (2) Existing systems derive reputation for domain names primarily *after* malicious activity is already underway, delaying detection. By the time existing reputation systems add a domain to a blacklist, the domain may have already been used in widespread abuse (*e.g.*, spam or phishing campaigns, drive-by downloads). (3) Because existing reputation systems cannot prevent malicious domain registrations, attackers continue to register new malicious domains even as their old domains become blacklisted, to sustain scam campaigns.

The ideal time to establish the reputation of DNS domains is thus *at registration time*, before the miscreants can profitably use them. Towards this goal, we design **PREDATOR** (Proactive Recognition and Elimination of Domain Abuse at Time-Of-Registration), a proactive reputation system that can accurately and automatically identify malicious domains at time-of-registration, rather than later at time-of-use. Such a proactive reputation can enable early detection of potentially malicious DNS domains, thus benefiting many stakeholders: (1) Network operators can take appropriate actions to protect their networks. For example, email servers can preemptively greylist [34] (*i.e.*, temporarily reject) emails that contain suspicious newly-registered domains and request re-delivery after a period (by which time the network operators can collect more evidence to make final decisions, such as examining the content that the domain hosts). (2) Registries or registrars can require more strict documentation or verification (*e.g.*, validation of payment instruments) before they approve registrations of domains with low predicted reputation. (3) Law enforcement and

CCS'16, October 24–28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978317

security professionals can prioritize investigation efforts and take down malicious sites early [51]. For example, in 2010, LegitScript[1] and registrar eNom have announced an agreement to devote efforts to taking down rogue Internet pharmacy domains [8].

PREDATOR is based on the intuition that, to make domain purchase as economical as possible miscreants must register large quantities of domains—typically in bulk—to ensure that they can remain agile as individual domains are blacklisted or taken down [33, 52]. The timing and targets of attacks may vary, but the domain registrations nonetheless exhibit telltale signatures, including the types of domains that they register and the temporal registration patterns they exhibit, that both distinguish them from benign registration behavior and are relatively non-malleable, due to the fact that attackers require access to large numbers of inexpensive domains to operate successfully.

Although a time-of-registration DNS domain reputation system offers many potential benefits, developing such a reputation system is difficult. Unlike other DNS reputation systems that can observe the ways a domain is used in practice (*e.g.*, by observing lookup patterns or content that the domain hosts), a reputation system that operates at registration time has much more limited information available for developing reputation. We identify features based on (1) the delegated registrars and the hosting infrastructure, (2) structural characteristics of the name itself, (3) previous registration history, and (4) correlations with registration bursts. Another challenge is the lack of full ground truth to evaluate performance; thus, our best hope is to compare the reputations that PREDATOR derives to those of existing blacklists. Of course, the reputation information in existing blacklists is not only late, it is also often incomplete. We also assess how PREDATOR can detect non-blacklisted but spam-related domains by sampling and manually checking those domains.

Our results show that PREDATOR can accurately determine the reputation of new domains with a low false positive rate (0.35%) and a good detection rate (70%). Although the performance is not perfect, *the benefits of establishing domain reputation at registration time are clear*: (1) PREDATOR can achieve early detection, often days or weeks before existing blacklists, which generally cannot detect domain abuse until an attack is already underway. The key advantage is to respond promptly for defense and limit the window during which miscreants might profitably use a domain. (2) As a first line of defense, PREDATOR can reduce the suspect domains to a pool of 3% of all new domains, while capturing 70% of the domains that will subsequently appear on well-known blacklists. Thus, our system enables prioritizing domains for subsequent, detailed analysis, and to find more malicious pages given a fixed amount of resources (*e.g.*, via URL crawlers or human-involved identification). (3) We show that PREDATOR can complement existing blacklists and capture additional domains hosting spam-related content that other blacklists miss.

We make the following contributions:

- We develop an approach to establish domain reputation based on the features evident at the time of domain registration, which provides early detection of potentially malicious domains.
- We identify and encode 22 features that help distinguish domain registration behavior characteristic of abuse from legitimate registration behavior, 16 of which have not been identified or analyzed in previous work.
- We incorporate these features into a state-of-the-art supervised learning algorithm and implement a prototype version of our proactive time-of-registration domain reputation system, PREDATOR.
- We perform a comprehensive empirical evaluation of PREDATOR using five months of logs of second-level .com and .net domain registrations, demonstrating its effectiveness for complementing existing blacklists.

[1]LegitScript is a service to verify and regulate online pharmacies, recognized by the National Association of Boards of Pharmacy.

2. BACKGROUND

The registration process of second-level domains involves three major participants: *registrants* (persons or companies that seek to acquire domain names), *registrars* (*e.g.*, GoDaddy), and *registries* (*e.g.*, Verisign). A registrant usually applies online to a registrar, which is an organization accredited by ICANN to contract with registries to sell domains. The registrar represents the registrant in submitting registration requests to the registry, which manages the relevant top-level domain (TLD) [25]. The registry allocates the domain name, adds the registration data to a central database, and publishes the DNS records in the top-level domain nameservers. The updates operate in close to real time [22, 23] and have a short interval between registration requests and domain activation in the zone. For every domain registration, the registrar charges the registrant and pays a fee to the associated registry. The annual cost for registrants to register a .com domain ranges from about $8.50 to $25 [45, 46]. The registry in turn pays a fee to ICANN for new domain names in the TLD zone.

The registration time periods range from one to ten years. Upon domain expiration, if registrants choose to renew, the domains remain in the zone. Otherwise, the domain expires, is removed from the zone, and becomes available for registration again. We categorize domains as *brand-new* (registered for the first time) or *re-registration* (the domain has previously appeared in the zone, and now a registrant registers it again after its expiration). We further characterize re-registration domains as *drop-catch* (re-registered immediately after its expiry) or *retread* (some time has passed since its previous removal from the zone). We can obtain information regarding expiring domains via registries or third-party sites [42, 59].

3. CASE STUDY: SPAMMER DOMAINS

As an illustration of how spammers exhibit distinctive registration behavior, consider the following instance at registrar Moniker. Figure 1 shows the counts of .com domain registrations from registrar Moniker on one day of March 2012. The x-axis gives the hour of the day (Eastern time). The y-axis shows the count of .com domain registrations for every five-minute epoch. The black bars indicate the counts of domains subsequently appearing on blacklists (including Spamhaus [49], URIBL [56], and a spam trap that we operate), while the white bars are the numbers of domains that are not blacklisted later. These instances provide insight into the characteristics of spammer domain registrations, such as: How many domains are registered together?; What do the names look like?; and How long do the existing blacklists take to block those domains? We briefly explore these questions using a case study.

(1) *Domain registrations occur in bursts*: Table 1 shows the detailed statistics of the five registration spikes in Figure 1. A five-minute epoch may have tens or even hundreds of domains registered for later spam activities. Presumably miscreants exploit the cheaper prices and management ease of bulk registration provided by registrars. For example, when registering over 100 .com domains, Moniker offers a 5% discount (as well as a 25% discount on privacy protection) [41], and GoDaddy provides a 36% discount (lowering the annual price from $12.99 to $8.29) [17]. Because miscreants likely often operate

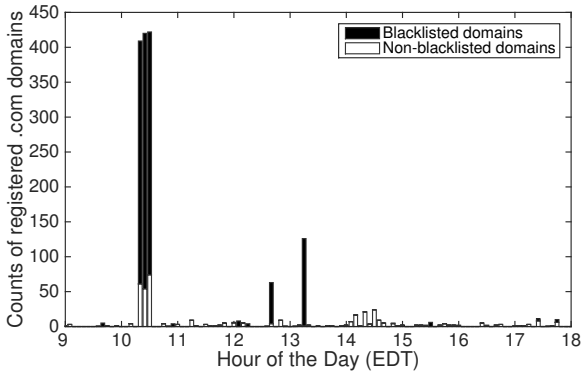


Figure 1: Domains registered by `Moniker` every 5 minutes on a day of March 2012. Each bar represents the count of .com domains registered per 5-minute epoch.

Five-minute epoch (EDT)	*Spammer domains*		*All domains*
	Brand-new	*Retread*	
10:15–10:20AM	0	**348**	409
10:20–10:25AM	0	**366**	420
10:25–10:30AM	0	**348**	422
12:35–12:40PM	**52**	7	63
1:10–1:15PM	**118**	7	126

Table 1: 5-minute epochs with spammer domain registrations from `Moniker`. The epochs correspond to the five registration spikes in Figure 1.

on thin margins and rely on volume for profitable operation, these discounts are significant.

(2) *Domains registered together are often at similar stages in the domain life-cycle*: Spammer domains in the same spike tend to exhibit the same life-cycle type, brand-new (*i.e.*, first-time registration) or retread (*i.e.*, re-registration with some time after previous expiration), as shown in Table 1. The registration spikes during 10:15–10:30AM (retread), and those during 12:35–1:15PM (brand-new) are likely from different attackers. This phenomenon indicates that in addition to making up names themselves, miscreants track domain names that are due for expiration in the `WHOIS` database (available from publicly released sources [42, 59]) and collect recently expired names in bulk.

(3) *Domains registered together may be similar to one another*: When we take a closer look at the spammer brand-new domains registered in the same epoch of 1:10–1:15PM, we observed that many names appeared similar to each other. Table 2 displays examples of names sharing the same substrings. We highlight the common substrings in bold. The names algorithmically generated by miscreants manifest characteristic lexical patterns.

Table 2 shows how many days pass until various spammer domains are blacklisted; in some cases, these delays may be several weeks, or even several months. Yet, the characteristic registration behaviors that we have outlined provide opportunities to detect and blacklist these domains earlier than existing techniques (*i.e.*, at registration time) because their registration behaviors are so distinctive. The following sections outline our general approach, as well as how we extract and encode the relevant features for establishing domain reputation.

4. PREDATOR ARCHITECTURE

We design PREDATOR to infer each domain's reputation immediately after registration. The decision process does not need to examine the Web content on the domains or wait until users are exposed to attacks. Our goal is for PREDATOR to act as a first layer of defense

Domains (highlight common strings)	*Blacklist delay*
ask**lender**home.com	92 days
askhome**lender**snow.com	51 days
ask**lender**shome.com	32 days
askhomes**lender**.com	24 days
askhome**lender**.com	12 days
askhome**lender**s.com	6 days
ask**lender**today.com	5 days
financilsart.com	122 days
financilss.com	71 days
financilsssky.com	19 days
financilsspro.com	18 days
financilspro.com	17 days
financilssart.com	9 days
financilssky.com	7 days
strokecarebeat.com	65 days
strokecaregreen.com	14 days
strokesoft.com	11 days

Table 2: Example of brand-new spammer domains registered in one single epoch (1:10–1:15PM EDT, per the bottom row of Table 1) from `Moniker`. The domain names with common strings are grouped and ordered by the blacklist delay.

to mitigate malicious URLs or domains that host spam-advertised sites.

Figure 2 shows how PREDATOR operates. We derive the domain registration information from zone updates. The *Domain Name Zone Alert* (*DNZA*) files contain changes to the zone, including (1) the addition of new domains, (2) the removal of existing domains, (3) changes to associated nameservers, and (4) changes to IP addresses of the nameservers. The DNZA files provide a real-time feed of domain registrations. The zone update data is recorded in five-minute intervals, which we define as *epochs*. The domains registered in the same epoch represent concurrent registrations and often share common properties. PREDATOR operates in two modes: an off-line *training mode* and an on-line *operation mode*, as we describe below.

Training mode. Based on the domain registration information, we extract three types of statistical features:

- *Domain profile features* (Section 5.1) focus on the current registration. The features can be derived from the public `WHOIS` information and the domain names.
- *Registration history features* (Section 5.2) are based on previous registration history. The features can be acquired from third-party services such as DomainTools [10], or they are available at registrars and registries.
- *Batch correlation features* (Section 5.3) examine the domains registered from the same registrar and within the same epoch. The information is available at registrars or registries.

We use prior knowledge to label a set of known spammer and non-spammer domains. With the labeled domains, the learning module takes the extracted features and uses a supervised learning technique to build a classifier. Section 6 describes the design of PREDATOR's classification approach in detail.

Operation mode. Upon a new domain registration, we extract the corresponding features and incorporate them into the classifier. The classifier assigns a reputation score by aggregating the weights learned in the training mode. If a domain is registered to launch malicious activities (*e.g.*, spam campaigns), we expect to assign a low reputation score. On the other hand, we want to assign a high score if a domain is for legitimate Internet services. If the score is lower than a threshold, we generate a detection report to flag the domain as malicious. Network operators or users can take advantage of the

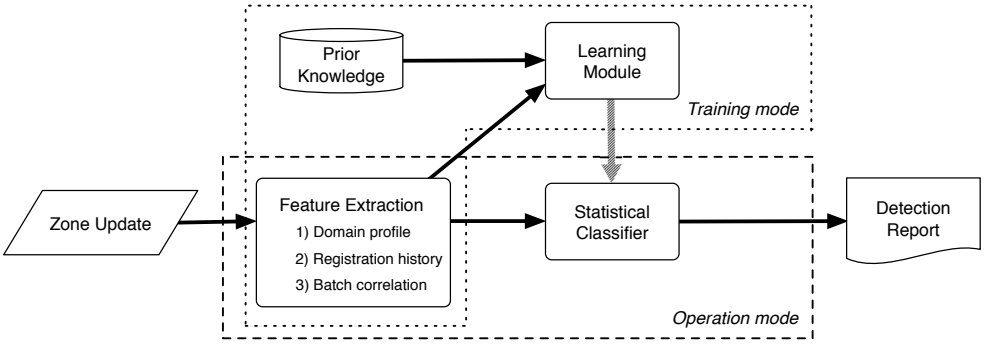


Figure 2: A high-level overview of PREDATOR**.**

early warning to limit the time that miscreants might profitably use a domain for malicious activities (in some cases, they may prevent the attacks from occurring in the first place).

5. IDENTIFYING CHARACTERISTIC FEATURES

We now expand on our earlier observations, exploring other characteristic domain registration behavior of miscreants and encoding these behaviors into features. We categorize the features into three groups: domain profile features, registration history features, and batch correlation features. Each feature is *categorical*, *continuous*, or *ordinal*. Categorical features typically arise when representing features that are nominal in nature. Such features do not present a meaningful notion of order, as values represent logically separate concepts. Continuous values naturally correspond to continuous-range type of features, and ordinal features denote the countable ordered ones (like integers). Most non-boolean features are of these two types.

Table 3 summarizes the features that PREDATOR uses. The first column shows the feature categories, the second describes the detailed features, the third indicates whether the features are categorical, continuous, or ordinal, and the last column lists previous work that has analyzed the feature. Out of 22 features, 16 have not been previously considered for registration-based detection; none have ever been incorporated into a classifier for time-of-registration reputation. We now describe each of these features in detail, discussing both the domain knowledge and intuition we used to identify the feature as well as how we ultimately encoded it. We observe similar patterns from .com and .net zones across different time periods (2012 and 2015, respectively).

5.1 Domain Profile Features

Domain profile features are those that we extract from the domain name or from the public `WHOIS` information regarding the current registration.

Registrar (categorical). End users need to select the delegated registrars to register domains. Miscreants presumably choose particular registrars due to the prices or policies from different ones. We find 79% of .com spammer domains were registered through only ten registrars. Similarly, 84% of .net spammer domains were attributed to ten registrars (five of the registrars overlapped). We map registrars to a group of binary features. A given dimension is set to 1 if and only if the corresponding registrar hosts the domain. There are 906 such categorical registrar features. Since a domain has a single designated registrar; only one such feature is set to 1 and the others are 0.

Authoritative nameservers (categorical). Without authoritative nameservers, people could not resolve the domains in the zone. The nameservers indicate how miscreants manage their domains (after choosing registrars). Miscreants may use self-resolving nameservers, where the nameserver is responsible for resolving its own domain (*e.g.*, the nameserver for `example.com` is `ns.example.com`, as opposed to `ns.third-party.com`) [13]. Although some other nameservers belong to major hosting companies that often host legitimate domains, they provide a finer-granularity indication of spammer domain registrations. The nameserver assignment usually happens close to domain registrations and might change with time. We collect authoritative nameservers for the domains within five minutes of domain registrations (*i.e.*, the epoch that we have defined) from the zone update files and map them to a set of binary features, where 1 means the nameserver resolves the domain. Because a domain could have multiple nameservers or nameserver changes, more than one attribute could have value 1.

Nameserver IP addresses and ASes (categorical). Multiple nameservers can resolve to the same IP address, and different IP addresses can originate from the same Autonomous Systems (ASes). Both IP addresses and ASes indicate underlying hosting infrastructure, which provides a means for identifying portions of the Internet with a higher prevalence of hosting spammer domains. We convert them to a group of binary attributes. Similar to the nameserver feature, more than one attribute could have a value of 1. In our experiment, we derive the IP addresses of the nameservers from the zone update files. For a newly registered domain, we obtain the nameservers within five minutes of the domain registration, and retrieve all IP addresses ever associated with the nameservers within one year before the domain registration. We use the mapping dataset from iPlane to map the IP addresses to Autonomous System numbers [29]. We focus on the IP addresses of the authoritative nameservers, available at registration time, while previous work mainly investigated IP addresses directly resolved for the domains [3, 5], usually configured later close to attack launch time (*e.g.*, using fast flux [24]).

Registration time (categorical). Miscreants need to repeatedly register domains for turnover in spam activities; this behavior may reveal certain temporal patterns. In `WHOIS` information, "Creation Date" indicates when a domain was registered. We have equivalent registration times from the zone update data, and extract hour-of-the-day and day-of-the-week. The hour of the day can reflect the time zones of registrants' locations, and the day of the week can capture the registrants' workflows. We extract the information according to Eastern Standard Time (*i.e.*, UTC -0500), although this choice is not significant because the purpose is to capture repeated temporal patterns of domain registrations. The hour of the day corresponds to 24 categorical features, and the day of the week maps to seven features.

Registration period (ordinal). A user can register a domain for one to ten years. "Expiration Date" in `WHOIS` shows when the domain

Category	*Feature*	*Type*	*New?*
Domain profile	Registrar	Categ.	[13, 20]
	Authoritative nameservers	Categ.	[13, 20]
	IP addresses of nameservers	Categ.	✓
	ASes of nameserver IP addresses	Categ.	✓
	Daily hour of registration	Categ.	✓
	Week day of registration	Categ.	✓
	Length of registration period	Ord.	✓
	Trigrams in domain name	Categ.	✓
	Ratio of the longest English word	Cont.	[20]
	Containing digits	Categ.	✓
	Containing "–"	Categ.	✓
	Name length	Ord.	✓
	Edit distances to known-bad domains	Cont.	✓
Registration history	Life cycle	Categ.	[20]
	Dormancy period for re-registration	Ord.	[20]
	Previous registrar	Categ.	✓
	Re-registration from same registrar	Categ.	✓
Batch correlation	Probability of batch size	Cont.	[20]
	Brand-new proportion	Cont.	✓
	Drop-catch proportion	Cont.	✓
	Retread proportion	Cont.	✓
	Name cohesiveness	Cont.	✓

Table 3: Summary of PREDATOR **features. We include references to previous work that has proposed similar features in the context of proactive registration-based detection. The "New?" column highlights features that this work explores for the first time; where a feature was previously studied or identified, we provide a reference.**

is about to expire. The user owns the domain and can renew it any time before its expiration. Longer registration terms mean higher up-front fees. We find that almost all spammer domains have one-year initial registration terms, presumably since spammers tend to abandon their domains due to blacklisting. We use the years between domain registration and its potential expiration as one feature.

Lexical patterns (containing multiple features). Registrants of benign domains tend to choose easy-to-remember names. On the other hand, to acquire large numbers of domains for attack campaigns, miscreants generate random-looking names or produce similar names by following certain rules. We compute the features to capture facets of the linguistic structure of domain names. Though some of these features are not strictly new, previous work has not investigated the lexical patterns of new domains across entire zones. When analyzing the naming patterns, we only use the name in the second-level domain and ignore the trailing TLD (like ".com").

1) *Trigram* (categorical). We use the standard trigram approach to represent a domain name and to examine the character sequence. A domain name can only consist of the characters of letters, digits, and hyphens [6, 40]. (Internationalized domain names use the same character set to encode Unicode characters [32].) Since DNS systems treat domain names in a case-insensitive manner, we convert the names into lower case to process. We have a group of $37^3 = 50,653$ binary features that represent all the possible trigrams on the allowed alphabet of 26 letters, 10 digits and the hyphen character. We set a given feature to 1 if and only if the corresponding trigram appears in the domain name; otherwise, we set it to 0.

2) *Ratio of the longest English word* (continuous). Miscreants may either generate pseudo-random names to avoid conflict with existing domains, or deliberately include readable words in the domain names to attract victim users to click and visit. We match the English words in a dictionary with a domain name to find the longest English word that the name contains. To normalize the feature, we compute the ratio of the length of the longest English word to the whole length of the name.

3) *Containing digits* (categorical). We observe that spammer domains (18% for .com and 42% for .net) are more likely to use numerical characters than non-spammer ones (10% for .com and 12% for .net). Possible reasons might be that miscreants add digits to derive numerous names from the same word affix or generate random names from a character set containing digits. We include a binary feature to indicate whether the domain name contains any digits.

4) *Containing "–"* (categorical). Similarly, miscreants can insert "–" to break individual words or concatenate multiple words, though our data did not show large differences regarding this attribute between spammer and non-spammer domain names. We include a binary feature to indicate whether the domain name contains any hyphens.

5) *Name length* (ordinal). Miscreants may create domains based on a specific template, such as random strings of a given length. We use the length (number of characters) of the domain name as a feature.

6) *Edit distances to known-bad domains* (continuous). We examine the similarity of a domain to a set of known-bad domains. We derive a set of previously known spammer domains based on the prior knowledge, compute the Levenshtein edit distances to the currently considered domain, and divide these edit distances by the length of the domain name for normalization. In our experiment (Section 7), we use the data from a separate month to extract known-bad domains, which do not overlap with any training or testing data. We take the five smallest normalized edit distances as features, which have values between 0 and 1 (we have experimented with various numbers of edit distances, and the detection performance remains similar). Although calculating edit distances is computationally expensive, it remains tractable given the pace of domain registrations. In our experiment, the size of the known-bad .com domain set is 66,598. On average, the calculation of edit distances for one domain to the set of known-bad domains requires 0.13 seconds on a 2.40GHz CPU machine; calculating these values for one month of new .com domains takes four days.

5.2 Registration History Features

Registration history features are based on previous registration history of a domain. If a domain has appeared in the zone before, we possess registration history, such as previous registrar and deletion time. Most of such features can be obtained from third-party services such as DomainTools [10] and Who.is [63], or they are available at registrars and registries. Due to the limitations of our data, we only consider the features regarding the most recent previous registration.

Life cycle (categorical). As mentioned in Section 2, we categorize domains as brand-new, drop-catch, and retread, depending on the registration history. Although the life-cycle type itself may not be a strong indicator whether the domain is registered for spam-related activity, it often provides discriminative information when combining with other features, such as the life-cycle types of the other domains registered from the same registrar and around the same time. In total, the life-cycle categories map to three binary features and only one of them is set to 1.

Dormancy period for re-registration (ordinal). The usual re-registration domains tend to include some that expired a long time ago. On the other hand, spammers intentionally collect expired domains from publicly released sources [42, 59], which concentrate on recently expired domains. We observe that 30% of non-spammer retread domains were re-registered within 90 days, while 65% of

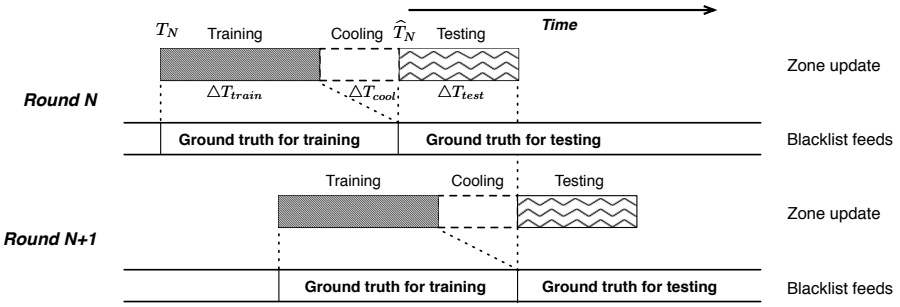


Figure 3: Sliding windows to train detection models.

spammer retread domains in .com and 37% of spammer retread domains in .net were re-registered within 90 days. We take the seconds between its last expiry and current registration as a feature. Regarding brand-new domains, the feature of dormancy period is not applicable, and we assign a default value (0 in our experiment).

Previous registrar (categorical). The previous registrar offers some insight from where and how spammers gather the expired domain information. We map previous registrars to a group of binary features. Only the feature corresponding to the previous registrar is set to 1, and the others have values 0. We handle brand-new domains whose previous registrar field is not applicable by simply adding a dummy registrar feature.

Re-registration from the same registrar (categorical). We add features to explicitly indicate whether the registrar of a re-registration domain is the same registrar of the previous registration. We observe that in the .com zone[2] 18% of non-spammer retread domains use the same previous registrar, while only 4% of spammer retread domains do so, presumably because miscreants choose particular registrars (Section 5.1), which are unlikely to be those used by prior legitimate registrants. We use a dummy value to deal with brand-new domains.

5.3 Batch Correlation Features

Batch correlation features examine domains within the same tuple ⟨registrar, five-minute epoch⟩, which we define as a batch. The batch information is initially known by registrars or registries.

Probability of batch size (continuous). Miscreants often register domains in large batches, presumably due to cheaper price of bulk registration or management ease. We identify the qualitatively different registration behavior by using the model of compound Poisson process, defined as in Hao et al. [20]. A low-probability batch size from the model indicates an abnormally large registration spike. We use the probability as a feature in our system.

Life-cycle proportion (continuous). As mentioned before, the registration history can characterize a domain as brand-new, drop-catch, or retread. Miscreants tend to register domains in a particular part of the domain life-cycle in a single batch due to how they select the names. We generate three features, each measuring the proportion of different life cycles for domains in the same batch. These three features sum to 1 by construction.

Name cohesiveness (continuous). Spammer domains registered in the same batch will sometimes have names lexically similar to one another, as miscreants use the same strategy or generation algorithm to produce a list of domains. To quantify the cohesiveness of the given domain name with respect to all other domain names in the same batch, we compute the edit distances of the domain to every other domain in the batch. We normalize these edit distances by dividing the length of the domain name to provide a similarity score. We then compute ten features as the numbers of domains with similarity between $[0, 0.1], [0, 0.2], \ldots, [0, 1.0]$. We use the logarithmic scale to account for the large variability of the batch sizes.

[2]Our data has no previous registrar information on .net, per Section 7.1.

6. CLASSIFIER DESIGN

This section introduces the Convex Polytope Machine (CPM), a supervised learning algorithm that we use (including our rationale for selecting this algorithm); the process of building the detection models; and the derivation of feature importance based on the models.

6.1 Supervised Learning: CPM

We want a classifier that can quickly train over large sets of data and achieve high accuracy. While linear Support Vector Machines (SVM) [12, 47] or comparable linear methods are often used in such high performance settings, nonlinearities in our data raise difficulties for SVM-style approaches. Instead, we employ a state-of-the-art supervised learning algorithm, the Convex Polytope Machine (CPM) [30]. CPM maintains an ensemble of linear sub-classifiers, and makes its final decision for incoming instances based on the maximum of all of their scores. More formally, suppose $\mathbf{x} \in \mathbb{R}^d$ is an instance of d features, and $\mathbf{w}^1, \ldots, \mathbf{w}^K \in \mathbb{R}^d$ represent the weights of the K sub-classifiers. We derive the score of $\mathbf{x}$ as:

$$f(\mathbf{x}) = \max_{1 \le k \le K} \langle \mathbf{x}, \mathbf{w}^k \rangle$$

The prediction score of $f(\mathbf{x})$ reflects how likely a domain is registered for spam-related activity. Geometrically, a CPM defines a convex polytope as the decision boundary to separate the two instance classes. In our application, it appears that this richer, non-linear decision boundary gives us high classification accuracy. Training of a CPM can be efficiently achieved by using the gradient descent technique [47]. To assess our design choice, we tested SVM [12] using `libsvm` with parameters tuned to our application. We found that in the low-false-positive region of operation, CPM produced a 10% higher true-positive rate, and trained faster than an SVM.

6.2 Building Detection Models

The first step of building the model is to normalize the continuous and ordinal features. We transform real values into the $[0, 1]$ interval to ensure that they do not overly dominate categorical features. We compute the ranges for each of the continuous and ordinal features to obtain max/min values, and normalize feature v to $(v - v_{min})/(v_{max} - v_{min})$. Since the categorical features are in binary, we do not need additional normalization process.

We adapt a sliding window mechanism for re-training models and evaluating the detection accuracy close to the real-deployment scenario. We define three windows: training $\triangle T_{train}$, cooling $\triangle T_{cool}$, and testing $\triangle T_{test}$. As shown in Figure 3, suppose at round N the training window starts at time T_N. The model will

be constructed at time $\widehat{T}_N = T_N + \triangle T_{train} + \triangle T_{cool}$, with the domains registered during $[T_N, T_N + \triangle T_{train}]$. Since this approach requires time to corroborate that a domain is indeed involved with spam-related activity, especially based on observations from blacklists, we use the ground truth collected during the period $[T_N, \widehat{T}_N]$ to label domains to build the model (in the training mode). In the testing period (corresponding to the operation mode), PREDATOR makes real-time prediction on those domains registered during $[\widehat{T}_N, \widehat{T}_N + \triangle T_{test}]$. The ground truth that we use in the testing period to evaluate the detection accuracy is composed of the domains showing on blacklists from time $\widehat{T}_N$ up to our last collection date of the blacklists. In the next round, $N + 1$, we move the time window forward by $\triangle T_{test}$, which makes the new model build at time $\widehat{T}_N + \triangle T_{test}$. The period $\triangle T_{test}$ indicates how frequently we re-train the model. Operators can customize the three window lengths according to different requirements and settings (see Section 7.4).

6.3 Assessing Feature Importance

Given a subset $S \subset \{1, \ldots, d\}$ of features, a derived CPM model $\{\mathbf{w}^1, \ldots, \mathbf{w}^K\}$, and a dataset of points $\{\mathbf{x}^1, \ldots, \mathbf{x}^n\}$, we derive a measure to evaluate the importance of the set of features in our classifier. If the model weights have large magnitudes while at the same time the associated features have low variance, *i.e.*, they are essentially constant, these dimensions are not particularly informative and should receive a low importance score. We design a scoring method to measure the total amount of variation on the score $f(\mathbf{x})$ over the dataset induced by the features S. In the case of a single linear classifier ($K = 1$), we measure this quantity as:

$$I_S^1 = \sqrt{\operatorname*{Var}_{\mathbf{x}} \left[\sum_{i \in S} \mathbf{w}_i^1 \mathbf{x}_i \right]}$$

To generalize this measure to the case $K \geq 2$, for each sub-classifier k, we compute the score I_S^k based on its subset of assigned instances A_k, and combine the scores.

$$I_S = \sqrt{\frac{|A_1|}{n} I_S^1 + \ldots + \frac{|A_K|}{n} I_S^K}$$
$$= \sqrt{\frac{1}{n} \sum_{k=1}^{K} |A_k| \operatorname*{Var}_{\mathbf{x} \in A_k} \left[\sum_{i \in S} \mathbf{w}_i^k \mathbf{x}_i \right]}$$

where A_k is composed of $\mathbf{x}$, satisfying $k = \arg\max_{k'} \langle \mathbf{w}^{k'}, \mathbf{x} \rangle$. Higher values of I_S indicate that the feature group S contributes more on the decision-making. We demonstrate the feature importance in Section 7.4.

7. EVALUATION

In this section, we report our evaluation results, compare the performance of PREDATOR to existing blacklists, and analyze the evasion scenarios.

7.1 Data Set and Labeling

Our primary dataset consists of changes made to the .com zone, the largest TLD [60], for a five-month period, March–July 2012. We obtain the DNZA files from Verisign (which have five-minute granularity), find the registrations of new domains, and extract the updates of authoritative nameservers and IP addresses. During March–July 2012, we have 12,824,401 newly registered second-level .com domains. To label the registered domains as legitimate or malicious, we collected public blacklisting information from March–October 2012 (eight months), including Spamhaus [49] (updated every 30 minutes), URIBL [56] (updated every 60 minutes), and a spam trap that we operate (real time). If a domain appeared on blacklists after registration, we label the domain as being involved in spam-related activities and registered by miscreants. To obtain benign labels, we queried McAfee SiteAdvisor [48] in June 2013 to find the domains that are reported as definitely benign. Eventually we have about 2% of .com domains with malicious labels and 4% with benign domain labels. We discuss the prediction results on labeled and unlabeled domains respectively in Section 7.2.

We also obtain the DNZA data of .net zone for five months, from October 2014 to February 2015, which contain 1,284,664 new domains. We used similar blacklists (URIBL and our spam trap) from October 2014 to May 2015 (eight months) to label malicious domains and queried McAfee in November 2015 to find benign labels. However, the information for .net domains is not complete, which just allows limited analysis on .net domains. We only have Spamhaus snapshot on December 7th 2015 (instead of a continuous feed), and the previous registrar information is not available.

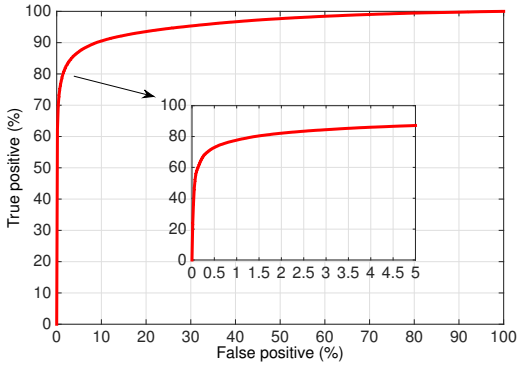


Figure 4: ROC of PREDATOR on .com domains. The inlay figure shows the ROC curve under the range of 0–5% false positives.

Page content of domain	*Percentage*
Advertisement links	38%
Lottery, survey, and coupon	7%
Adult content	7%
Merchandise	7%
Pharmaceutical	6%
Download of Software and files	5%
Online gambling	4%
No obvious spam-related content	26%

Table 4: Breakdown of manually checking 100 random samples of unlabeled domains that PREDATOR classifies as malicious. (Note that pages with no obvious spam-related content might still host other malicious activities such as drive-by downloads.)

7.2 Detection Accuracy

We demonstrate the accuracy of PREDATOR in terms of false positive rate, which is the ratio of benign domains misclassified as malicious to all benign instances; and detection rate, which accounts for the ratio of correctly predicted spammer domains to all spammer domain samples. By setting different thresholds, we make tradeoffs between false positive rates and detection rates.

For .com domains, we use data from March 2012 to extract the known-bad domain set and derive probability models for registration batches, and take April–July 2012 for our experiments. We used the sliding window method (introduced in Section 6.2) and tested different window lengths, where better results resulted from longer training windows (*i.e.*, more domains for training) and shorter testing windows

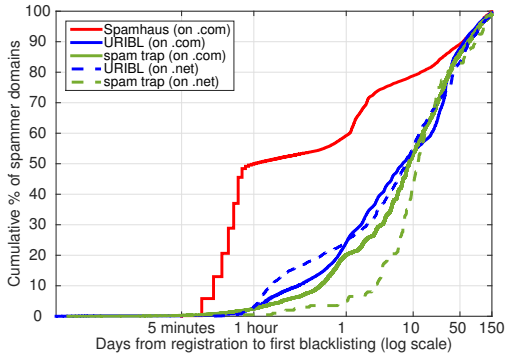


Figure 5: Distribution of days between domain registration and appearance in either our spam trap, URIBL, or Spamhaus, which indicates how early PREDATOR **can make detections compared to the existing blacklists (no Spamhaus timing information on .net).**

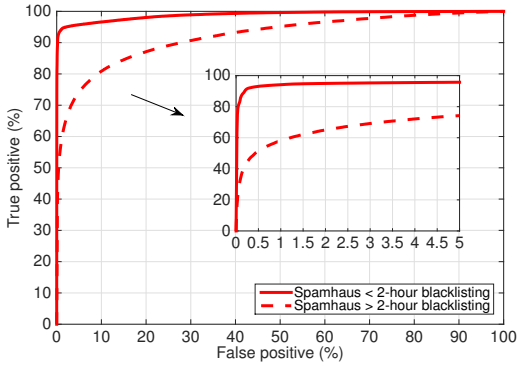


Figure 6: ROC of PREDATOR **using domains that Spamhaus blacklisted within the first 2 hours of registration and after the first 2 hours of registration for labels.**

Training window \ *Testing window*	7 days	35 days	56 days
35 days	**70.00%**	68.29%	66.81%
21 days	67.10%	64.96%	60.56%
14 days	64.13%	60.51%	58.22%

Table 5: Detection rates (under a 0.35% false positive rate) with different window settings. With shorter training windows and longer testing windows (*i.e.*, less frequent re-training), the prediction will become more inaccurate. Our experiments show that the performance is not overly sensitive to the window settings.

(*i.e.*, more frequent re-training). We demonstrate the performance results of PREDATOR with the setting of the training window to 35 days, the cooling window to 1 day, and the testing window to 7 days, which produces good detection accuracy and allows realistic operation (see Section 7.4 for detailed discussion on window selection).

Figure 4 shows the ROC curve of PREDATOR. The x-axis shows the false positive rate, and the y-axis shows the detection rate. The inlay figure shows the ROC curve for the range of 0–5% false positives. PREDATOR achieves good detection rates under low false positives. For example, with a 70% detection rate, the false positive is 0.35%. We emphasize that these results only rely on features constructed from the limited information available at registration time. Thus, as an early-warning mechanism, PREDATOR can effectively detect many domains registered for malicious activities.

Results on the entire .com zone. We project the 0.35% false positive to the entire .com zone. Since there are around 80,000 new domains everyday, the daily false positives are about 280 domains (as an upper estimate, assuming all domains are benign). Given that even the known spammer domains totalled more than 1,700 every day, PREDATOR can greatly help to narrow down the set of suspect domains. We ran additional tests to examine how many unlabeled domains are classified as spam-related by using the constructed detection model on the entire zone dataset, about seven million .com new domains registered over three months. With a threshold under a 0.35% false positive rate (in Figure 4, obtaining a 70% detection rate), PREDATOR reports about 1,000 unlabeled domains per day as spam-related, the same magnitude as the labeled spammer domains (1,700 per day). Overall, PREDATOR predicts 3% of all newly registered .com domains as malicious, which capture 70% of the malicious domains showing up later on blacklists. As a first line of defense, PREDATOR can effectively reduce and prioritize suspect domains for further inspection (*e.g.*, URL crawling or manual investigation) and find more malicious pages given a fixed amount of resources. In Section 7.3, we investigate to what extent the unlabeled domains that PREDATOR classifies as malicious indeed connect to illicit online activities while missed by current blacklists.

Detection accuracy on .net domains. We performed a similar experiment to report the detection accuracy on .net zone (five months in 2014–2015). Due to data limitation, two features, the previous registrar and re-registration from the same registrar, are unavailable, and in the blacklists Spamhaus only has a single snapshot. With the same sliding window setting, the detection rate on .net domains is 61% (close to the 70% on .com domains) under a 0.35% false positive rate. The result shows that PREDATOR can successfully make prediction at different zones. In the rest experiments, we focus on .com domains (.net domains either yield similar results or cannot conduct the analysis due to data defect).

7.3 Comparison to Existing Blacklists

We investigate and compare different blacklists and find that PREDATOR can help to mitigate the shortcomings of current blacklisting methods and detect malicious domains earlier.

Detection of more spammer domains. The first property we examine is *completeness*, which explores how many spammer domains PREDATOR detects compared to other blacklists. We find that during May–July 2012, the exclusive blacklisted .com domains (*i.e.*, not reported by other feeds) on Spamhaus, URIBL, and our spam trap number 24,015, 4,524, and 442 respectively. Each blacklist has many domains not identified by other sources, which indicates the existing blacklists are not perfect to detect all malicious domains. Having incomplete blacklists makes it quite challenging to develop more accurate registration-time detection, and also shows how PREDATOR can complement current detection methods by leveraging the central observation of domain registrations.

To investigate the potential of PREDATOR to detect spammer domains that existing blacklists miss, we sample the unlabeled .com domains that PREDATOR predicts as bad, and manually check whether they host any spam-related content. When we performed the analysis, most of the domains had expired, and we could not directly crawl and assess their content. Alternatively, we used historical snapshots from DomainTools [10] and the Internet Archive [28]. Table 4 shows the page categories of 100 randomly sampled domains. 74% of the unlabelled domains that PREDATOR predict as bad refer to content

Rank	*Category*	*Feature*	*Score ratio*
1	**D**	Authoritative nameservers	100.00%
2	**D**	Trigrams in domain name	64.88%
3	**D**	IP addresses of nameservers	62.98%
4	**D**	Registrar	61.28%
5	**D**	ASes of nameserver IP addresses	30.80%
6	**D**	Daily hour of registration	30.30%
7	**B**	Name cohesiveness	28.98%
8	**D**	Weekday of registration	22.58%
9	**R**	Dormancy period for re-registration	20.58%
10	**R**	Re-registration from same registrar	19.50%
11	**R**	Life cycle	18.55%
12	**D**	Edit distances to known-bad domains	17.72%
13	**R**	Previous registrar	16.50%
14	**B**	Brand-new proportion	14.60%
15	**B**	Retread proportion	13.71%
16	**B**	Drop-catch proportion	12.90%
17	**D**	Containing digits	11.25%
18	**D**	Name length	10.71%
19	**D**	Ratio of the longest English word	9.60%
20	**B**	Probability of batch size	8.66%
21	**D**	Containing "–"	8.02%
22	**D**	Length of registration period	3.34%

Table 6: Ranking of feature importance in PREDATOR **(D for domain profile category, R for registration history category, and B for batch correlation category).**

that is often hosted on spam-related sites (*e.g.*, pharmaceutical content, adult content), and 26% of these pages have no obvious spam-related content (though might have other malicious activities that we cannot measure, such as drive-by downloads). This result demonstrates PREDATOR's ability to augment existing blacklists by exposing malicious domains that they fail to report.

Early detection. Another important blacklisting characteristic concerns *delay*: how long after a spammer domain registration the blacklists identify it. Detection delays leave users unprotected in the interim, allowing attackers to reap greater benefits from their domains. Figure 5 shows the distribution of the time between a .com/.net domain's registration and its first appearance on blacklist (no Spamhaus timing information on .net). We observe that both URIBL and our spam trap take significant time to identify spammer domains, *e.g.*, around 50% of blacklisted domains manifest after 7 days. Clearly, PREDATOR can make detection early, even weeks before appearance on blacklists, which provides more time to respond or prevent attacks. On the other hand, Spamhaus has a mode of time-of-registration blacklisting, where a certain amount of blacklisting occurs shortly after domain registrations. We use a two-hour threshold to estimate conservatively, since the Spamhaus feed that we use updates every half hour.

To assess the degree to which Spamhaus uses time-of-registration features to blacklist domains, and to explore how the features that Spamhaus uses compare to our features, we evaluate the accuracy of PREDATOR using (for both training and testing) only the domains that Spamhaus blacklists in the first two hours of registration to label malicious domains. We then repeat the analysis for domains that Spamhaus blacklists more than two hours of registration. Figure 6 shows the prediction accuracy of PREDATOR using these two sets of labels. PREDATOR achieves a detection rate above 93% with a 0.35% false positive when using as labels the domains that were blacklisted within two hours. The high accuracy result suggests PREDATOR features already contain most of those used by Spamhaus (anecdotes indicate that Spamhaus involves only simple features, and in Section 7.4 we further infer what features Spamhaus relies on). PREDATOR also achieves decent accuracy using the domains that

Rank	*Category*	*Feature*	*Score ratio*
1	**D**	Authoritative nameservers	100.0%
2	**D**	Registrar	47.72%
3	**D**	IP addresses of nameservers	44.26%
4	**D**	Trigrams in domain name	37.91%
5	**D**	ASes of nameserver IP addresses	24.98%
6	**D**	Daily hour of registration	14.23%
7	**R**	Re-registration from same registrar	19.50%
8	**B**	Retread proportion	11.48%
9	**R**	Life cycle	10.93%
10	**B**	Drop-catch proportion	10.70%

Table 7: Top 10 ranked features in PREDATOR **when applying on Spamhaus $<$ 2-hour blacklisting (same categories as in Table 6).**

Spamhaus blacklists more than two hours after registration as labels: a 0.35% false positive for a detection rate of about 47%. This result shows PREDATOR can detect some malicious domains much faster than Spamhaus can.

7.4 Analysis of the Classifier

Contribution of new features. As shown in Section 5 (and Table 3), 16 out of 22 features that we identified and incorporated into PREDATOR are proposed for the first time. To evaluate the contribution of these new features to improving the accuracy, we run an experiment with solely the features that previous work has explored, and the detection rate drops to 58.40% (under a 0.35% false positive rate). On the other hand, PREDATOR with the full feature set achieves the 70% detection rate under the same false positive rate. The result shows that the new features that we introduced can considerably improve the detection accuracy. Our work is the first to develop a reputation system that is able to accurately and automatically predict the maliciousness of a domain at registration time. Note that prior research either just presented preliminary measurement results [20], or was limited to extrapolating from particular properties, such as self-resolving nameservers [13].

Sliding window settings. We have used the sliding window mechanism (introduced in Section 6.2) in the experiments to simulate the practical deployment scenario of PREDATOR. The length of the training window determines how much data to build the classifier, and the length of the testing window indicates how often we re-train the model. In Table 5, we compare the detection rates with different training/testing windows under a 0.35% false positive rate (for our data, different cooling windows yield similar performance and we set it to one day). Shorter training windows (*i.e.*, less training data) and longer testing windows (*i.e.*, less frequent re-training) will produce less accurate predictions. The results show that the accuracy of PREDATOR is not overly sensitive to the window settings. For our data, training on 35-day data and re-training weekly is sufficient to maintain accuracy. We expect that when PREDATOR runs on a different dataset, additional analysis should be performed and the window settings may vary across different datasets.

Feature ranking. We use the scoring method in Section 6.3 to rank our features. The scores represent how much the features can contribute to identify either malicious or benign labels. For easy interpretation, we calculate the score ratio by dividing the score values with the largest one. Table 6 ranks all registration-based features on .com zone (with the most important feature at top). The capitalized letters in the second column indicates the feature categories: **D** for domain profile, **R** for registration history, and **B** for batch correlation. Seven of the top ten features belong to the domain profile category. This result is quite encouraging, since most of these features can be

collected with less overhead and from public sources, such as WHOIS database.

The ranking of features can help us to infer what features Spamhaus appears to rely on for its time-of-registration blacklisting. Table 7 lists the feature importance when we apply our detection algorithm on the domains blacklisted by Spamhaus within two hours of registration. We focus on the top ten features. The difference between the ratio for the first two features in Table 7 appears larger than the ratio difference in Table 6, which indicates that Spamhaus was inclined to use the feature of authoritative nameservers for detection. In fact, when considering only the nameservers that have more than 90% of their hosted domains appearing on Spamhaus time-of-registration blacklisting, those nameservers account for 86% of all domains appearing on Spamhaus time-of-registration blacklisting. The observation suggests that Spamhaus heavily uses nameservers to make time-of-registration blacklisting decisions.

7.5 Evasion

As with any detection system, sophisticated attackers may attempt to evade PREDATOR. We argue that trying to evade PREDATOR will alter the economics for miscreants to acquire domains and consequently impair their attack capability.

We first consider two groups of PREDATOR features, nameserver-related and lexical, which have relatively high ranks in Table 6. Nameserver-related features include nameservers of the second-level domains and the corresponding IP addresses and ASes (rank 1, 3, 5 in Table 6). These features are inherent to the hosting infrastructure, which require effort for attackers to alter. We evaluate PREDATOR's performance under different evasion scenarios by excluding the corresponding features from the system, as shown in Figure 7. The red solid curve corresponds to the ROC curve of PREDATOR incorporating all features on .com zone, and the blue dashed curve indicates the ROC curve if miscreants evade nameserver-related patterns (three features). Though PREDATOR's performance degrades, it still achieves a good level of detection accuracy. This observation suggests that nameserver-related features are important, but in their absence other features can still contribute to retain good detection.

We next consider lexical features (rank 2, 7, 12, 17–19, 21 in Table 6). Generating a large number of names is not a trivial task, as the plausibility of the names could influence an attack's efficacy. Changing naming patterns to use irrelevant words or random strings could reduce click-through rates for spam or phishing. Attackers may attempt to exploit HTML emails or pages to manipulate the displayed domains/URLs. However, the mismatch between the hyperlink text and the underlying domains would make it easier to be detected by previous work [15]. If miscreants try to evade name-similarity by inserting numerical or hyphen characters, PREDATOR's features of names containing digits and "–" can capture this. The green dashed curve in Figure 7 indicates the ROC with nameserver-related and lexical features excluded (six more features). We observe that detection accuracy further weakens, suggesting that lexical features help reduce false positives.

The batch features (rank 14, 15, 16, 20 in Table 6), despite their relatively low ranks, can contribute to the detection accuracy. In Figure 7, the purple dashed curve shows the ROC curve with batch features further excluded (four more features), where the performance decreases, especially in the low-false-positive area. As we will discuss later, if miscreants attempt to evade PREDATOR by mimicking legitimate behavior (including changing the patterns in registration batches), this would impair their attack capability, from financial and volume perspectives.

Registrars represent an essential feature in our system (rank 4 in Table 6) to capture miscreants' tactics. Miscreants tend to use the registrars that are cheaper and more tolerant of their activities [37]. Evading this feature forces attackers to change to less "scam-friendly" registrars. Even if miscreants switch to different sets of registrars, PREDATOR can over time automatically learn the shifts and detect new malicious domains. The black dotted curve in Figure 7 illustrates the performance of using only the registrar feature. The relatively large decrease in accuracy suggests that single features have limited detection power, and combinations of other features can significantly improve effectiveness.

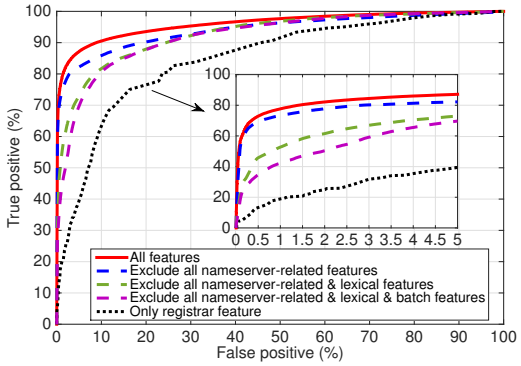


Figure 7: ROC of PREDATOR under different features (simulated evasion scenarios).

Financial cost of evasion. Evading some PREDATOR features forces miscreants to spend more to acquire domains. As discussed, to evade registrar feature (rank 4), attackers have to use some registrars with higher prices that are not their first choice. Evading the bulk-registration feature (rank 20) forces attackers to spend more by foregoing bulk discounts. Some miscreants also pay with stolen credit cards (to reduce cost and avoid tracing their real identities) [11, 14], which requires bulk registration, since fraud detection disables cards after several purchases. Evading the registration-period-length feature (rank 22) likewise requires greater expense due to paying in advance for multi-year terms. Verisign charges a $7.85 annual fee to registrars for each .com domain registration [58]. Miscreants can get low prices close to this amount from registrars affiliated with scam activities (*e.g.*, ABSystems [50]) or registrars offering cheap prices/discounts. If miscreants switch to the largest registrar GoDaddy, with an annual price of $12.99 [17]; register in small batches without taking any bulk discount; and commit to a 2-year registration term then the price per domain rises $\frac{\$12.99}{\$7.85} \times 2 \approx 3.3$ times of the price that miscreants originally pay.

Evading by decreasing volume. Evading some of the PREDATOR features constrains the volume of names that miscreants can easily register. In an attempt to evade the life-cycle proportion features (rank 14–16), miscreants may mix different life-cycle types of domains in the same registration batch. However, this will require incorporating multiple methods to generate domain names, increasing management effort. Moreover, we observe that spammer domains possess a lower brand-new proportion (66%) than non-spammer domains (77%). Given the same quantity of generated brand-new names, miscreants need to reduce re-registration domains to mimic the life-cycle proportions of general domains. We have $\frac{66\%}{100\%-A\%} = 77\%$. Solving for A, this yields that miscreants must alter 14% of their domain registrations to simulate benign behaviors. Note that this estimate demonstrates the impact to evade a single feature. To change other behaviors, such as the aforementioned lexical or registrar features, miscreants may have to further cut their domain registrations to small volumes. Another feature is the dormancy period for re-registration domains (rank 9). Domains re-registered by miscreants are often

those that expired more recently, presumably because miscreants actively mine expired domains. Evading the dormancy period feature forces miscreants to wait longer to register expired domains, which in turn limits the domains miscreants can use over a given period.

In general terms, evasion attempts considerably increase economic and management costs for attackers; PREDATOR raises the bar for miscreants to acquire and profitably use the domains. The combination of the features is effective to detect malicious domains. Altering one or two features will not significantly aid miscreants to fly under the radar.

8. DISCUSSION

In this section, we discuss possible deployment scenarios, as well as some limitations of our work.

Deployment scenarios. Network operators and security practitioners can benefit from PREDATOR in the following ways. (1) Network operators can take appropriate actions to protect their networks and users. For example, email servers can greylist [34] (*i.e.*, temporarily reject) emails that PREDATOR predicts as suspicious and request the originating servers to try again after a period. Legitimate senders are expected to resend the emails, while spammers usually do not properly handle retries [36]. Meanwhile, network operators can collect more evidence before retry attempts to make final decisions, such as examining the Web content on the domain. (2) Registries or registrars can require more strict documentation or verification (*e.g.*, validation of payment instruments or inquiring the domain purpose), before they approve registrations of domains with low reputation scores. Mitigating domain abuse is consistent with the registrars' responsibilities under the ICANN's Registrar Accreditation Agreement [26] (some registrars have taken important roles [8, 21]), and it can also help to identify and deter the illegal registrations with stolen credit cards [11, 14] (which cause loss from registrars, including refunds and chargeback fees). (3) Law enforcement and cyber-security professionals can prioritize their investigations (for time/resource-consuming analysis, *e.g.*, crawling the page content or repeated manual investigation by an analyst) and proactively monitor low-reputation domains, since the domains selected by PREDATOR are more likely to prove malicious. (4) Operators could also incorporate PREDATOR into other detection systems (*e.g.*, spam filters, botnet detection systems) by using it to provide an additional "confidence score" of registration to help them determine whether a particular domain appears malicious.

Limitations and future work. Given that domain registrations under a single zone provide centralized observation opportunities, miscreants may register domains across different TLDs, especially with the expansion of large numbers of new TLDs [27]. Thus, one direction for future work is incorporating cross-zone features into the classification model. Although PREDATOR is somewhat resistant to evasion, designing a more robust system against the attackers' continual attempts to mislead or evade the classifier is a promising area for future work. While our approach achieves good accuracy, for higher detection rates the false positive rates increase as well. Another area for improvement is combining post-registration detection techniques, such as DNS monitoring or Web crawling, to develop hierarchical decision-making mechanisms for higher accuracy.

9. RELATED WORK

We compare previous work on analyzing and detecting domains and URLs that are used in illicit online activities.

DNS-based detection. Most previous DNS-based detection studies focused on analyzing lookup traffic. Notos [3] and EXPOSURE [5] leverage traffic from local recursive DNS servers to establish domain reputations. Gao et al. used temporal correlation in DNS queries to detect malicious domain groups [16]. Various previous work has analyzed DNS traffic to detect fast-flux domains [24, 44], or malware domains exploiting resource records (*e.g.*, TXT records) as the communication channels [9, 31, 66]. Other work inspects DNS traffic close to top-level domain servers to detect abnormal activity [4, 19]. In contrast, PREDATOR derives domain reputation using registration features to enable early detection, without monitoring DNS traffic.

Registration and domain market. Recent research has paid attention to the registrars, registries, and the domain market, including domain-name speculation, typosquatting, and domain parking [1, 2, 7, 54, 61]. Liu et al. found that registry policy changes and registrar-level takedown had at least temporary effects in deterring spam-advertised domains [37]. Felegyhazi et al. investigated registration information to extrapolate malicious domains from specific instances of known-bad domains, primarily relying on the properties of DNS servers [13]. Hao et al. measured and modeled domain registrations of spammer domains [20]. While their work only presents preliminary measurement results and does not examine how to leverage the findings for detection, it hints at the potential of building a registration-based reputation system. We designed PREDATOR just for that purpose, to accurately detect spammer domains at time-of-registration, and our work studied significantly more features.

Website and URL detection. A conventional technique to detect malicious Web pages is through automatic URL crawling tools. The detection can be based on the page content [43, 57], the presence of cloaking and redirection [35, 62], or the link structure leading to the pages [64]. Thomas et al. built a large-scale system to crawl URLs in email and Twitter feeds to detect malicious messages [55]. Our study is orthogonal to Web crawling methods, and does not require page visits. The output of PREDATOR can help with prioritizing what suspect sites to crawl and inspect. A related approach is to use various lexical and host-based features of the URL for detection, but exclude Web page content [38, 39, 65]. These detection methods require waiting until miscreants use the URLs for attacks. Our work, on the other hand, provides proactive detection of domains before malicious URLs propagate on the Internet.

10. CONCLUSION

Because determining the reputation of DNS domains can significantly aid in defending against many Internet attacks, establishing DNS domain name reputation as quickly as possible provides major benefits. Whereas existing DNS reputation systems establish domain reputation based on features evident after the domain is in use, PREDATOR can accurately establish domain reputation at the time of domain registration, before domains ever see use in attacks. Our results show that PREDATOR can provide more accurate and earlier detection compared to existing blacklists, and significantly reduce the number of suspicious domains requiring more resource-intensive or time-consuming inspection.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. We also thank Christopher Kruegel, Kevin Borgolte, and Jennifer Rexford for many helpful suggestions and discussions to improve the paper. This work was supported in part by the National Science Foundation awards CNS-1237265, CNS-1535796, CNS-1540066, and by a gift from Google. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] P. Agten, W. Joosen, F. Piessens, and N. Nikiforakis. Seven Months' Worth of Mistakes: A Longitudinal Study of Typosquatting Abuse. In *Network and Distributed System Security Symposium (NDSS)*, Feb. 2015.

[2] S. Alrwais, K. Yuan, E. Alowaisheq, Z. Li, and X. Wang. Understanding the Dark Side of Domain Parking. In *23rd USENIX Security Symposium*, Aug. 2014.

[3] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *19th USENIX Security Symposium*, Aug. 2010.

[4] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, and D. Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *20th USENIX Security Symposium*, Aug. 2011.

[5] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *Network and Distributed System Security Symposium (NDSS)*, Feb. 2011.

[6] R. Braden. *Requirements for Internet Hosts – Application and Support*. Internet Engineering Task Force, Oct. 1989. RFC 1123.

[7] S. E. Coull, A. M. White, T.-F. Yen, F. Monrose, and M. K. Reiter. Understanding Domain Registration Abuses. In *25th International Information Security Conference*, Sept. 2010.

[8] Demand Media. eNom and LegitScript LLC Announce Agreement to Identify Customers Operating Illegal Online Pharmacies. http://www.businesswire.com/news/home/20100921005657/en/, 2010.

[9] C. J. Dietrich, C. Rossow, F. C. Freiling, H. Bos, M. van Steen, and N. Pohlmann. On Botnets that use DNS for Command and Control. In *European Conference on Computer Network Defense*, Sept. 2011.

[10] DomainTools. http://www.domaintools.com, 2015.

[11] S. Ellis. Business Email Compromise Scams on the Rise. http://www.markmonitor.com/mmblog/business-email-compromise-scams/, 2015. MarkMonitor Blog.

[12] R. Fan, K. Chang, C. Hsieh, X. Wang, and Lin. LIBLINEAR : A Library for Large Linear Classification. *The Journal of Machine Learning Research*, 9(2008):1871–1874, 2008.

[13] M. Felegyhazi, C. Kreibich, and V. Paxson. On the Potential of Proactive Domain Blacklisting. In *3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, Apr. 2010.

[14] P. Festa. Identity Thieves Strike eBay. http://www.cnet.com/news/identity-thieves-strike-ebay/, 2012. CNET.

[15] I. Fette, N. Sadeh, and A. Tomasic. Learning to Detect Phishing Emails. In *16th International Conference on World Wide Web (WWW)*, May 2007.

[16] H. Gao, V. Yegneswaran, Y. Chen, P. Porras, S. Ghosh, J. Jiang, and H. Duan. An Empirical Reexamination of Global DNS Behavior. In *ACM SIGCOMM*, Aug. 2013.

[17] Godaddy Bulk Registration Prices. http://www.godaddy.com/domains/searchbulk.aspx, 2014.

[18] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: The Underground on 140 Characters or Less. In *17th ACM Conference on Computer and Communications Security (CCS)*, Oct. 2010.

[19] S. Hao, N. Feamster, and R. Pandrangi. Monitoring the Initial DNS Behavior of Malicious Domains. In *ACM Internet Measurement Conference (IMC)*, Nov. 2011.

[20] S. Hao, M. Thomas, V. Paxson, N. Feamster, C. Kreibich, C. Grier, and S. Hollenbeck. Understanding the Domain Registration Behavior of Spammers. In *ACM Internet Measurement Conference (IMC)*, Oct. 2013.

[21] K. J. Higgins. Google, GoDaddy Help Form Group To Fight Fake Online Pharmacies. http://www.darkreading.com/d/d-id/1134946, 2010. Dark Reading.

[22] S. Hollenbeck. *VeriSign Registry Registrar Protocol Version 2.0.0*. Internet Engineering Task Force, Nov. 2003. RFC 3632.

[23] S. Hollenbeck. *Extensible Provisioning Protocol*. Internet Engineering Task Force, Aug. 2009. RFC 5730.

[24] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measureing and Detecting Fast-Flux Service Networks. In *Network and Distributed System Security Symposium (NDSS)*, Feb. 2008.

[25] IANA. Root Zone Database. http://www.iana.org/domains/root/db, 2016.

[26] ICANN. Registrar Accreditation Agreement (Section 3.18). https://www.icann.org/resources/pages/approved-with-specs-2013-09-17-en, 2013.

[27] ICANN. Delegated Strings of New TLDs. http://newgtlds.icann.org/en/program-status/delegated-strings, 2015.

[28] Internet Archive. http://archive.org, 2015.

[29] iPlane. http://iplane.cs.washington.edu/data/data.html, 2015.

[30] A. Kantchelian, M. C. Tschantz, P. L. B. Ling Huang, A. D. Joseph, and J. D. Tygar. Large-Margin Convex Polytope Machine. In *Neural Information Processing Systems (NIPS)*, Dec. 2014.

[31] A. M. Kara, H. Binsalleeh, M. Mannan, A. Youssef, and M. Debbabi. Detection of Malicious Payload Distribution Channels in DNS. In *Communication and Information Systems Security Symposium*, June 2014.

[32] J. Klensin. *Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework*. Internet Engineering Task Force, Aug. 2010. RFC 5890.

[33] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamcraft: An Inside Look At Spam Campaign Orchestration. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, Apr. 2009.

[34] M. Kucherawy and D. Crocker. *Email Greylisting: An Applicability Statement for SMTP*. Internet Engineering Task Force, June 2012. RFC 6647.

[35] N. Leontiadis, T. Moore, and N. Christin. Measuring and Analyzing Search-Redirection Attacks in the Illicit Online Prescription Drug Trade. In *20th USENIX Security Symposium*, Aug. 2011.

[36] P. Lieven, B. Scheuermann, M. Stini, and M. Mauve. Filtering Spam Email Based on Retry Patterns. In *IEEE International Conference on Communications (ICC)*, June 2007.

[37] H. Liu, K. Levchenko, M. Felegyhazi, C. Kreibich, G. Maier, G. M. Voelker, and S. Savage. On the Effects of Registrar-level Intervention. In *4th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, Mar. 2011.

[38] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. In *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, June 2009.

[39] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying Suspicious URLs: An Application of Large-Scale Online Learning. In *26th International Conference on Machine Learning (ICML)*, June 2009.

[40] P. V. Mockapetris. *Domain Names - Concepts and Facilities*. Internet Engineering Task Force, Nov. 1987. RFC 1034.

[41] Moniker Bulk Registration Discounts. http://www.moniker.com/service/discounts, 2014.

[42] NameJet Domain Name Aftermarket. http://www.namejet.com/pages/downloads.aspx, 2015.

[43] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting Spam Web Pages through Content Analysis. In *15th International Conference on World Wide Web (WWW)*, May 2006.

[44] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *25th Annual Computer Security Applications Conference (ACSAC)*, Dec. 2009.

[45] Annual Price $8.49 for .com at INTERNET.bs. http://internetbs.net/, 2015.

[46] Annual Price $24.95 for .com at DomainPeople. http://www.domainpeople.com/domain-names/pricing.html, 2015.

[47] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *24th International Conference on Machine Learning (ICML)*, June 2007.

[48] McAfee SiteAdvisor. https://www.siteadvisor.com/.

[49] Spamhaus. http://www.spamhaus.org/.

[50] Spamhaus. ABSystems Domain Registrar De-accredited. http://www.spamhaus.org/rokso/evidence/ROK10342/, 2013.

[51] Spamhaus. Can Registrars Suspend Domains for Spam and Abuse? https://www.spamhaus.org/faq/section/Generic%20Questions#127, 2015.

[52] B. Stone-Gross, R. Abman, R. Kemmerer, C. Kruegel, D. Steigerwald, and G. Vigna. The Underground Economy of Fake Antivirus Software. In *10th Workshop on Economics of Information Security (WEIS)*, June 2011.

[53] Symantec. Rise in URL Spam. http://www.symantec.com/connect/blogs/rise-url-spam, 2013.

[54] J. Szurdi, B. Kocso, G. Cseh, J. Spring, M. Felegyhazi, and C. Kanich. The Long "Taile" of Typosquatting Domain Names. In *23rd USENIX Security Symposium*, Aug. 2014.

[55] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and Evaluation of a Real-Time URL Spam Filtering Service. In *32rd IEEE Symposium on Security and Privacy*, May 2011.

[56] URIBL. http://www.uribl.com/.

[57] T. Urvoy, E. Chauveau, P. Filoche, and T. Lavergne. Tracking Web Spam with HTML Style Similarities. *ACM Transactions on the Web*, 2(1):3:1–3:28, 2008.

[58] Verisign. Verisign Announces Increase in .com/.net Domain Name Fees. https://investor.verisign.com/releasedetail.cfm?releaseid=591560, 2011.

[59] Verisign Domain Countdown. http://domaincountdown.verisignlabs.com, 2011.

[60] Verisign. The Domain Name Industry Brief. http://www.verisign.com/assets/domain-name-brief-dec2012.pdf, 2012.

[61] T. Vissers, W. Joosen, and N. Nikiforakis. Parking Sensors: Analyzing and Detecting Parked Domains. In *Network and Distributed System Security Symposium (NDSS)*, Feb. 2015.

[62] D. Y. Wang, S. Savage, and G. M. Voelker. Cloak and Dagger: Dynamics of Web Search Cloaking. In *18th ACM Conference on Computer and Communications Security (CCS)*, Oct. 2011.

[63] Who.is. http://who.is/domain-history, 2015.

[64] B. Wu and B. D. Davison. Identifying Link Farm Spam Pages. In *14th International Conference on World Wide Web (WWW)*, May 2005.

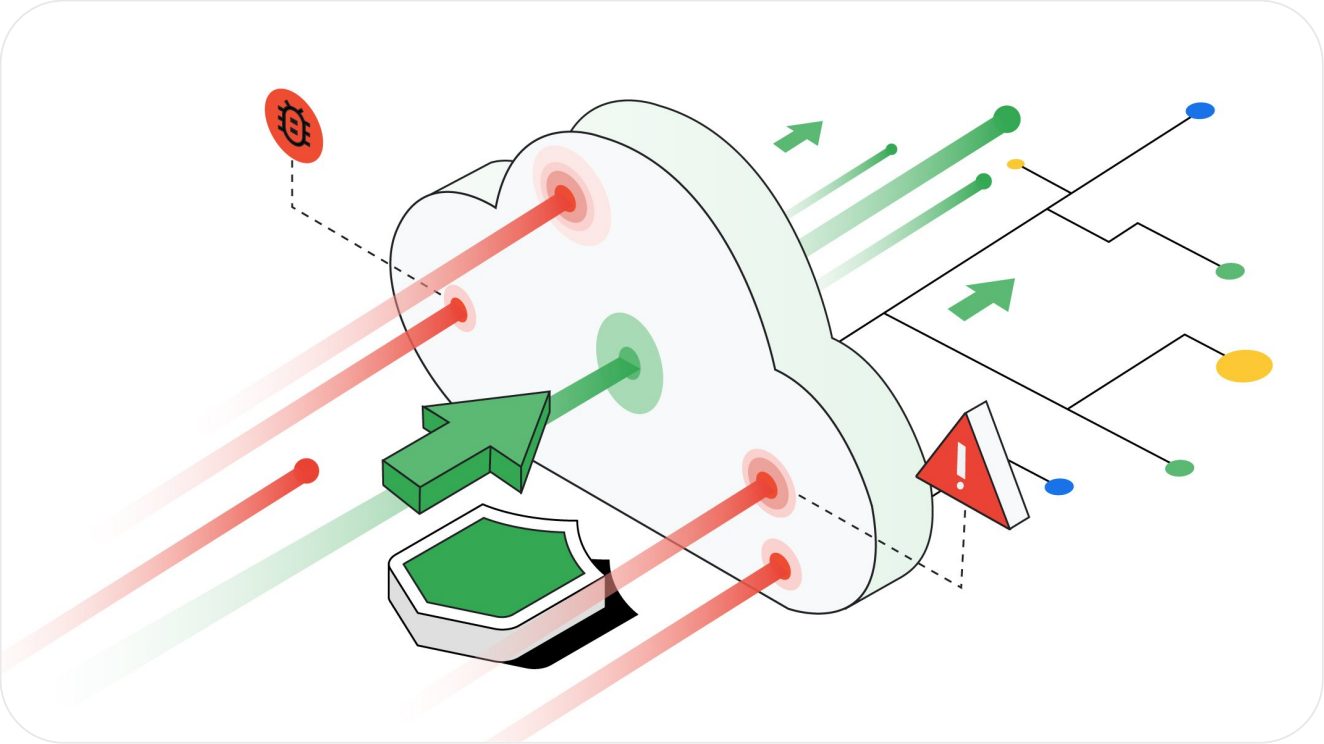
[65] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipko. Spamming Botnets: Signatures and Characteristics. In *ACM SIGCOMM*, Aug. 2008.

[66] K. Xu, P. Butler, and D. Yao. DNS for Massive-Scale Command and Control. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 10(3):143–153, 2013.

Security & Identity

How it works: The novel HTTP/2 'Rapid Reset' DDoS attack

October 10, 2023



Daniele Iamartino

Juho Snellman

A number of Google services and Cloud customers have been targeted with a novel HTTP/2-based DDoS attack which peaked in August. These attacks were significantly larger than any previously-reported Layer 7 attacks, with the largest attack surpassing 398 million requests per second.

The attacks were largely stopped at the edge of our network by Google's global load balancing infrastructure and did not lead to any outages. While the impact was minimal, Google's DDoS Response Team reviewed the attacks and added additional protections to further mitigate similar attacks. In addition to Google's internal response, we helped lead a coordinated disclosure process with industry partners to address the new HTTP/2 vector across the ecosystem.

Below, we explain the predominant methodology for Layer 7 attacks over the last few years, what changed in these new attacks to make them so much larger, and the mitigation strategies we believe are effective against this attack type. This article is written from the perspective of a reverse proxy architecture, where the HTTP request is terminated by a reverse proxy that forwards requests to other services. The same concepts apply to HTTP servers that are integrated into the application server, but with slightly different considerations which potentially lead to different mitigation strategies.

A primer on HTTP/2 for DDoS

Since late 2021, the majority of Layer 7 DDoS attacks we've observed across Google first-party services and Google Cloud projects protected by Cloud Armor have been based on HTTP/2, both by number of attacks and by peak request rates.

efficient for legitimate clients can also be used to make DDoS attacks more efficient.

Stream multiplexing

HTTP/2 uses "streams", bidirectional abstractions used to transmit various messages, or "frames", between the endpoints. “Stream multiplexing” is the core HTTP/2 feature which allows higher utilization of each TCP connection. Streams are multiplexed in a way that can be tracked by both sides of the connection while only using one Layer 4 connection. Stream multiplexing enables clients to have multiple in-flight requests without managing multiple individual connections.

One of the main constraints when mounting a Layer 7 DoS attack is the number of concurrent transport connections. Each connection carries a cost, including operating system memory for socket records and buffers, CPU time for the TLS handshake, as well as each connection needing a unique four-tuple, the IP address and port pair for each side of the connection, constraining the number of concurrent connections between two IP addresses.

In HTTP/1.1, each request is processed serially. The server will read a request, process it, write a response, and only then read and process the next request. In practice, this means that the rate of

includes the network latency, proxy processing time and backend request processing time. While HTTP/1.1 pipelining is available in some clients and servers to increase a connection's throughput, it is not prevalent amongst legitimate clients.

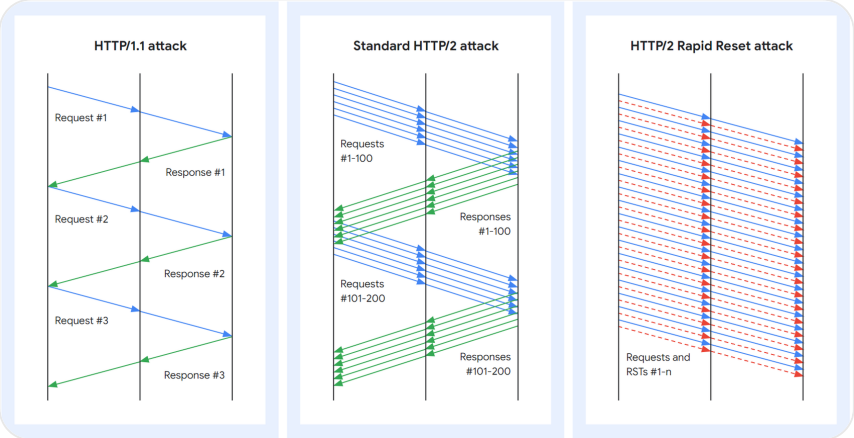
With HTTP/2, the client can open multiple concurrent streams on a single TCP connection, each stream corresponding to one HTTP request. The maximum number of concurrent open streams is, in theory, controllable by the server, but in practice clients may open 100 streams per request and the servers process these requests in parallel. It's important to note that server limits can not be unilaterally adjusted.

For example, the client can open 100 streams and send a request on each of them in a single round trip; the proxy will read and process each stream serially, but the requests to the backend servers can again be parallelized. The client can then open new streams as it receives responses to the previous ones. This gives an effective throughput for a single connection of 100 requests per round trip, with similar round trip timing constants to HTTP/1.1 requests. This will typically lead to almost 100 times higher utilization of each connection.

The HTTP/2 Rapid Reset attack

sending a RST_STREAM frame. The protocol does not require the client and server to coordinate the cancellation in any way, the client may do it unilaterally. The client may also assume that the cancellation will take effect immediately when the server receives the RST_STREAM frame, before any other data from that TCP connection is processed.

This attack is called Rapid Reset because it relies on the ability for an endpoint to send a RST_STREAM frame immediately after sending a request frame, which makes the other endpoint start working and then rapidly resets the request. The request is canceled, but leaves the HTTP/2 connection open.



HTTP/1.1 and HTTP/2 request and response pattern

The HTTP/2 Rapid Reset attack built on this capability is simple: The client opens a large number of streams at once as in the standard HTTP/2 attack, but rather than waiting for a response to each request stream from the server or proxy, the client cancels each request immediately.

in flight. By explicitly canceling the requests, the attacker never exceeds the limit on the number of concurrent open streams. The number of in-flight requests is no longer dependent on the round-trip time (RTT), but only on the available network bandwidth.

In a typical HTTP/2 server implementation, the server will still have to do significant amounts of work for canceled requests, such as allocating new stream data structures, parsing the query and doing header decompression, and mapping the URL to a resource. For reverse proxy implementations, the request may be proxied to the backend server before the RST_STREAM frame is processed. The client on the other hand paid almost no costs for sending the requests. This creates an exploitable cost asymmetry between the server and the client.

Another advantage the attacker gains is that the explicit cancellation of requests immediately after creation means that a reverse proxy server won't send a response to any of the requests. Canceling the requests before a response is written reduces downlink (server/proxy to attacker) bandwidth.

HTTP/2 Rapid Reset attack variants

In the weeks after the initial DDoS attacks, we have seen some Rapid Reset attack variants. These

standard HTTP/2 DDoS attacks.

The first variant does not immediately cancel the streams, but instead opens a batch of streams at once, waits for some time, and then cancels those streams and then immediately opens another large batch of new streams. This attack may bypass mitigations that are based on just the rate of inbound RST_STREAM frames (such as allow at most 100 RST_STREAMs per second on a connection before closing it).

These attacks lose the main advantage of the canceling attacks by not maximizing connection utilization, but still have some implementation efficiencies over standard HTTP/2 DDoS attacks. But this variant does mean that any mitigation based on rate-limiting stream cancellations should set fairly strict limits to be effective.

The second variant does away with canceling streams entirely, and instead optimistically tries to open more concurrent streams than the server advertised. The benefit of this approach over the standard HTTP/2 DDoS attack is that the client can keep the request pipeline full at all times, and eliminate client-proxy RTT as a bottleneck. It can also eliminate the proxy-server RTT as a bottleneck if the request is to a resource that the HTTP/2 server responds to immediately.

RFC 9113, the current HTTP/2 RFC, suggests that an attempt to open too many streams should invalidate

servers will not process those streams, and is what enables the non-cancelling attack variant by almost immediately accepting and processing a new stream after responding to a previous stream.

A multifaceted approach to mitigations

We don't expect that simply blocking individual requests is a viable mitigation against this class of attacks — instead the entire TCP connection needs to be closed when abuse is detected. HTTP/2 provides built-in support for closing connections, using the GOAWAY frame type. The RFC defines a process for gracefully closing a connection that involves first sending an informational GOAWAY that does not set a limit on opening new streams, and one round trip later sending another that forbids opening additional streams.

However, this graceful GOAWAY process is usually not implemented in a way which is robust against malicious clients. This form of mitigation leaves the connection vulnerable to Rapid Reset attacks for too long, and should not be used for building mitigations as it does not stop the inbound requests. Instead, the GOAWAY should be set up to limit stream creation immediately.

This leaves the question of deciding which connections are abusive. The client canceling

request processing. Typical situations are when a browser no longer needs a resource it had requested due to the user navigating away from the page, or applications using a long polling approach with a client-side timeout.

Mitigations for this attack vector can take multiple forms, but mostly center around tracking connection statistics and using various signals and business logic to determine how useful each connection is. For example, if a connection has more than 100 requests with more than 50% of the given requests canceled, it could be a candidate for a mitigation response. The magnitude and type of response depends on the risk to each platform, but responses can range from forceful GOAWAY frames as discussed before to closing the TCP connection immediately.

To mitigate against the non-cancelling variant of this attack, we recommend that HTTP/2 servers should close connections that exceed the concurrent stream limit. This can be either immediately or after some small number of repeat offenses.

Applicability to other protocols

We do not believe these attack methods translate directly to HTTP/3 (QUIC) due to protocol

Despite that, our recommendation is for HTTP/3 server implementations to proactively implement mechanisms to limit the amount of work done by a single transport connection, similar to the HTTP/2 mitigations discussed above.

Industry coordination

Early in our DDoS Response Team's investigation and in coordination with industry partners, it was apparent that this new attack type could have a broad impact on any entity offering the HTTP/2 protocol for their services. Google helped lead a coordinated vulnerability disclosure process taking advantage of a pre-existing coordinated vulnerability disclosure group, which has been used for a number of other efforts in the past.

During the disclosure process, the team focused on notifying large-scale implementers of HTTP/2 including infrastructure companies and server software providers. The goal of these prior notifications was to develop and prepare mitigations for a coordinated release. In the past, this approach has enabled widespread protections to be enabled for service providers or available via software updates for many packages and solutions.

During the coordinated disclosure process, we reserved CVE-2023-44487 to track fixes to the various HTTP/2 implementations.

The novel attacks discussed in this post can have significant impact on services of any scale. All providers who have HTTP/2 services should assess their exposure to this issue. Software patches and updates for common web servers and programming languages may be available to apply now or in the near future. We recommend applying those fixes as soon as possible.

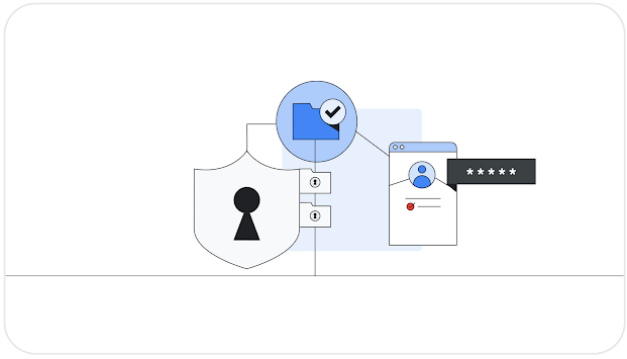
For our customers, we recommend patching software and enabling the Application Load Balancer and Google Cloud Armor, which has been protecting Google and existing Google Cloud Application Load Balancing users.

Posted in Security & Identity—Networking—Google Cloud

Related articles

Security & Identity

Cloud CISO Perspectives: Why ISACs are valuable security



Security & Identity

What's new in Assured Workloads: Japan regions, move

HTTP/2 Rapid Reset: deconstructing the record-breaking attack

10/10/2023

 Lucas Pardue  Julien Desgats

16 min read

This post is also available in 简体中文, 繁體中文, 日本語, 한국어, Deutsch, Français and Español.

Starting on Aug 25, 2023, we started to notice some unusually big HTTP attacks hitting many of our customers. These attacks were detected and mitigated by our automated DDoS system. It was not long however, before they started to reach

record breaking sizes — and eventually peaked just above 201 million requests per second. This was nearly 3x bigger than our previous biggest attack on record.

Under attack or need additional protection? Click here to get help.

Concerning is the fact that the attacker was able to generate such an attack with a botnet of merely 20,000 machines. There are botnets today that are made up of hundreds of thousands or millions of machines. Given that the entire web typically sees only between 1–3 billion requests per second, it's not inconceivable that using this method could focus an entire web's worth of requests on a small number of targets.

Detecting and Mitigating

This was a novel attack vector at an unprecedented scale, but Cloudflare's existing protections were largely able to absorb the brunt of the attacks. While initially we saw some impact to customer traffic — affecting roughly 1% of requests during the initial wave of attacks — today we've been able to refine our mitigation methods to stop the attack for any Cloudflare customer without it impacting our systems.

We noticed these attacks at the same time two other major industry players — Google and AWS — were seeing the same. We worked to harden Cloudflare's systems to ensure that, today, all our customers are protected from this new DDoS attack method without any customer impact. We've also participated with Google and AWS in a coordinated disclosure of the attack to impacted vendors and critical infrastructure providers.

This attack was made possible by abusing some features of the HTTP/2 protocol and server implementation details (see CVE-2023-44487 for details). Because the attack abuses an underlying weakness in the HTTP/2 protocol, we believe any

vendor that has implemented HTTP/2 will be subject to the attack. This included every modern web server. We, along with Google and AWS, have disclosed the attack method to web server vendors who we expect will implement patches. In the meantime, the best defense is using a DDoS mitigation service like Cloudflare's in front of any web-facing web or API server.

This post dives into the details of the HTTP/2 protocol, the feature that attackers exploited to generate these massive attacks, and the mitigation strategies we took to ensure all our customers are protected. Our hope is that by publishing these details other impacted web servers and services will have the information they need to implement mitigation strategies. And, moreover, the HTTP/2 protocol standards team, as well as teams working on future web standards, can better design them to prevent such attacks.

RST attack details

HTTP is the application protocol that powers the Web. HTTP Semantics are common to all versions of HTTP — the overall architecture, terminology, and protocol aspects such as request and response messages, methods, status codes, header and trailer fields, message content, and much more. Each individual HTTP version defines how semantics are transformed into a "wire format" for exchange over the Internet. For example, a client has to serialize a request message into binary data and send it, then the server parses that back into a message it can process.

HTTP/1.1 uses a textual form of serialization. Request and response messages are exchanged as a stream of ASCII characters, sent over a reliable transport layer like TCP, using the following format (where CRLF means carriage-return and linefeed):

```
HTTP-message   = start-line CRLF
                 *( field-line CRLF )
```

```
CRLF
[ message-body ]
```

For example, a very simple GET request for `https://blog.cloudflare.com/` would look like this on the wire:

```
GET / HTTP/1.1 CRLFHost: blog.cloudflare.comCRLFCRLF
```

And the response would look like:

```
HTTP/1.1 200 OK CRLFServer: cloudflareCRLFContent-Length:
100CRLFtext/html; charset=UTF-8CRLFCRLF<100 bytes of data>
```

This format **frames** messages on the wire, meaning that it is possible to use a single TCP connection to exchange multiple requests and responses. However, the format requires that each message is sent whole. Furthermore, in order to correctly correlate requests with responses, strict ordering is required; meaning that messages are exchanged serially and can not be multiplexed. Two GET requests, for `https://blog.cloudflare.com/` and `https://blog.cloudflare.com/page/2/`, would be:

```
GET / HTTP/1.1 CRLFHost: blog.cloudflare.comCRLFCRLFGET /page/2/ HTTP/1.1
CRLFHost: blog.cloudflare.comCRLFCRLF
```

With the responses:

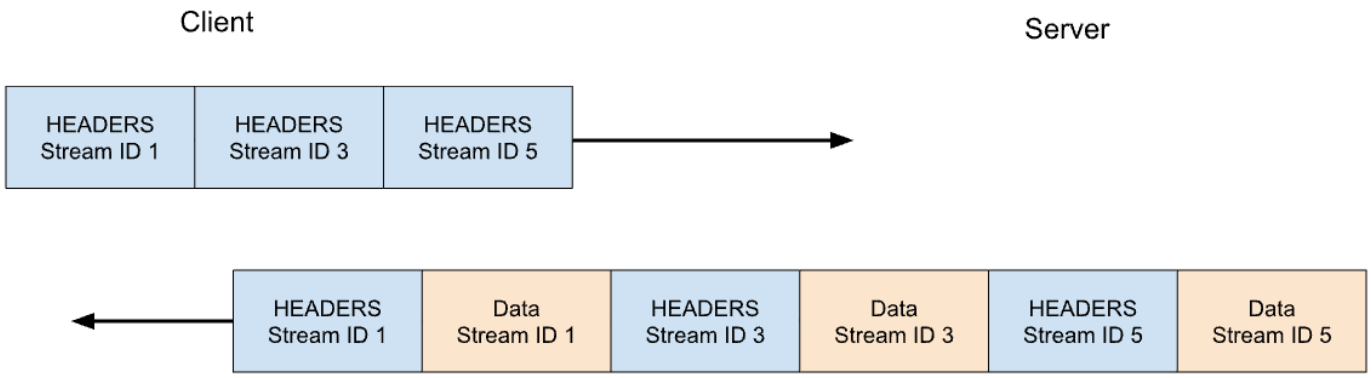
```
HTTP/1.1 200 OK CRLFServer: cloudflareCRLFContent-Length:
100CRLFtext/html; charset=UTF-8CRLFCRLF<100 bytes of data>CRLFHTTP/1.1
200 OK CRLFServer: cloudflareCRLFContent-Length: 100CRLFtext/html;
charset=UTF-8CRLFCRLF<100 bytes of data>
```

Web pages require more complicated HTTP interactions than these examples. When visiting the Cloudflare blog, your browser will load multiple scripts, styles and media assets. If you visit the front page using HTTP/1.1 and decide quickly to

navigate to page 2, your browser can pick from two options. Either wait for all of the queued up responses for the page that you no longer want before page 2 can even start, or cancel in-flight requests by closing the TCP connection and opening a new connection. Neither of these is very practical. Browsers tend to work around these limitations by managing a pool of TCP connections (up to 6 per host) and implementing complex request dispatch logic over the pool.

HTTP/2 addresses many of the issues with HTTP/1.1. Each HTTP message is serialized into a set of **HTTP/2 frames** that have type, length, flags, stream identifier (ID) and payload. The stream ID makes it clear which bytes on the wire apply to which message, allowing safe multiplexing and concurrency. Streams are bidirectional. Clients send frames and servers reply with frames using the same ID.

In HTTP/2 our GET request for `https://blog.cloudflare.com` would be exchanged across stream ID 1, with the client sending one HEADERS frame, and the server responding with one HEADERS frame, followed by one or more DATA frames. Client requests always use odd-numbered stream IDs, so subsequent requests would use stream ID 3, 5, and so on. Responses can be served in any order, and frames from different streams can be interleaved.



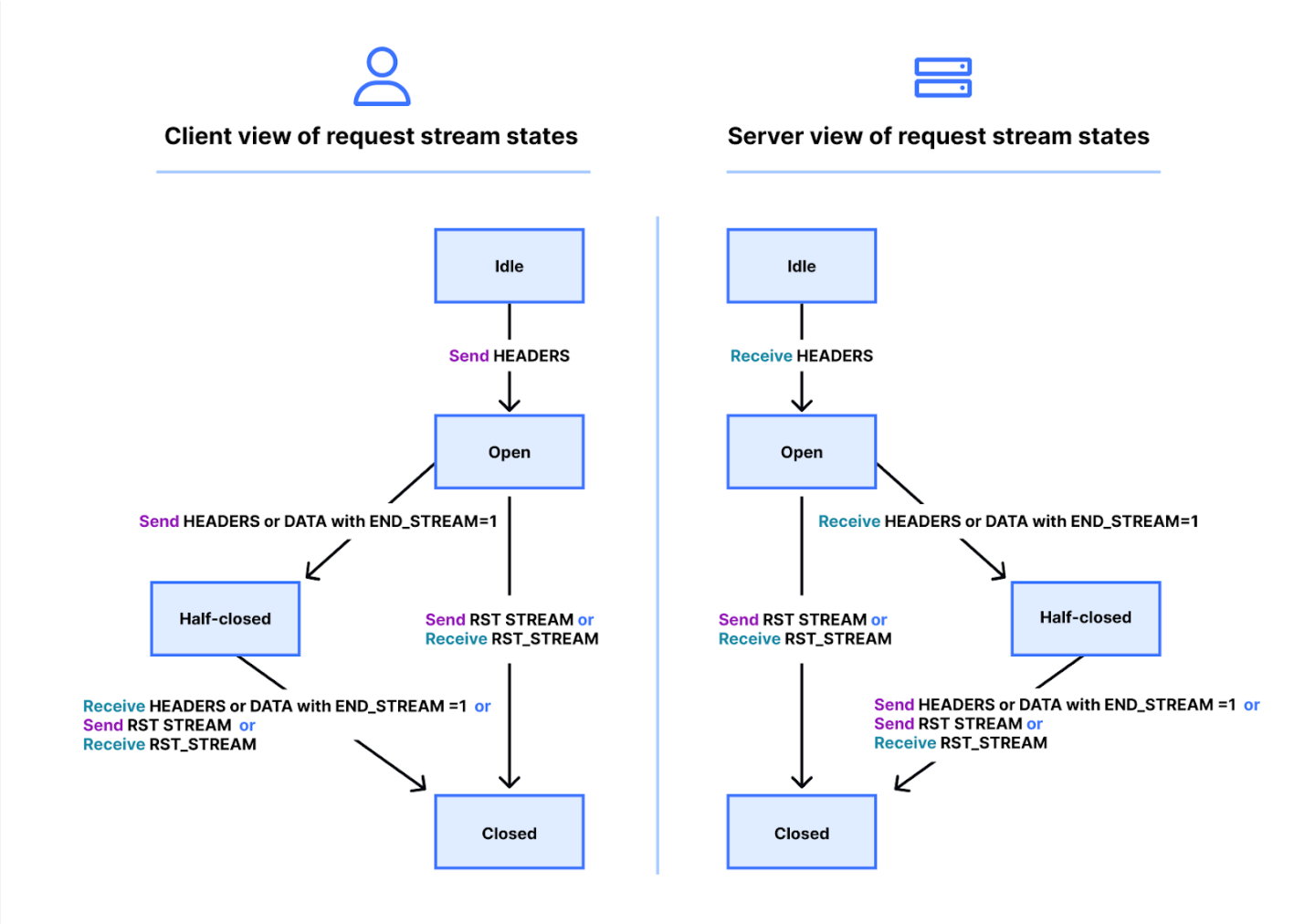
Stream multiplexing and concurrency are powerful features of HTTP/2. They enable more efficient usage of a single TCP connection. HTTP/2 optimizes resources fetching especially when coupled with prioritization. On the flip side,

making it easy for clients to launch large amounts of parallel work can increase the peak demand for server resources when compared to HTTP/1.1. This is an obvious vector for denial-of-service.

In order to provide some guardrails, HTTP/2 provides a notion of maximum active concurrent streams. The SETTINGS_MAX_CONCURRENT_STREAMS parameter allows a server to advertise its limit of concurrency. For example, if the server states a limit of 100, then only 100 requests can be active at any time. If a client attempts to open a stream above this limit, it must be rejected by the server using a RST_STREAM frame. Stream rejection does not affect the other in-flight streams on the connection.

The true story is a little more complicated. Streams have a lifecycle. Below is a diagram of the HTTP/2 stream state machine. Client and server manage their own views of the state of a stream. HEADERS, DATA and RST_STREAM frames trigger transitions when they are sent or received. Although the views of the stream state are independent, they are synchronized.

HEADERS and DATA frames include an END_STREAM flag, that when set to the value 1 (true), can trigger a state transition.



Let's work through this with an example of a GET request that has no message content. The client sends the request as a HEADERS frame with the END_STREAM flag set to 1. The client first transitions the stream from **idle** to **open** state, then immediately transitions into **half-closed** state. The client half-closed state means that it can no longer send HEADERS or DATA, only WINDOW_UPDATE, PRIORITY or RST_STREAM frames. It can receive any frame however.

Once the server receives and parses the HEADERS frame, it transitions the stream state from idle to open and then half-closed, so it matches the client. The server half-closed state means it can send any frame but receive only WINDOW_UPDATE, PRIORITY or RST_STREAM frames.

The response to the GET contains message content, so the server sends

HEADERS with END_STREAM flag set to 0, then DATA with END_STREAM flag set to 1. The DATA frame triggers the transition of the stream from **half-closed** to **closed** on the server. When the client receives it, it also transitions to closed. Once a stream is closed, no frames can be sent or received.

Applying this lifecycle back into the context of concurrency, HTTP/2 states:

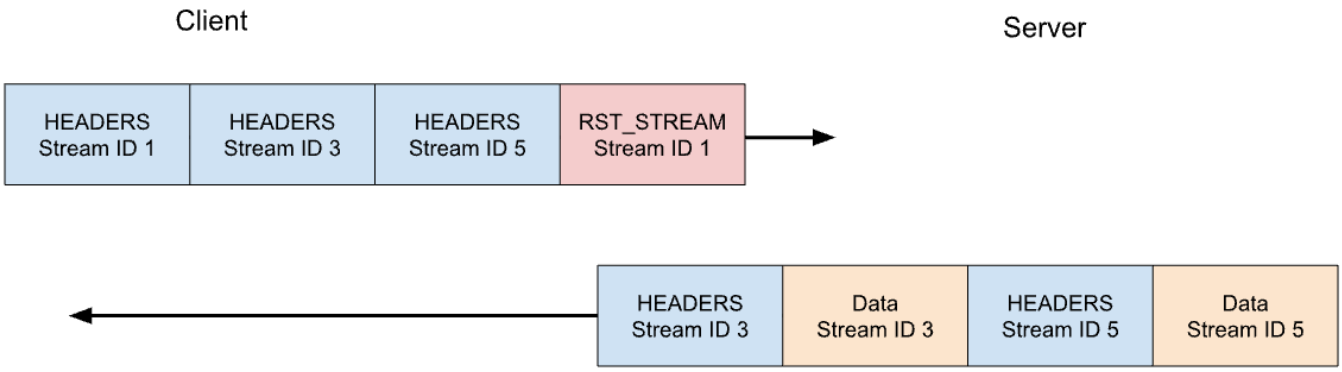
Streams that are in the "open" state or in either of the "half-closed" states count toward the maximum number of streams that an endpoint is permitted to open. Streams in any of these three states count toward the limit advertised in the SETTINGS_MAX_CONCURRENT_STREAMS setting.

In theory, the concurrency limit is useful. However, there are practical factors that hamper its effectiveness— which we will cover later in the blog.

HTTP/2 request cancellation

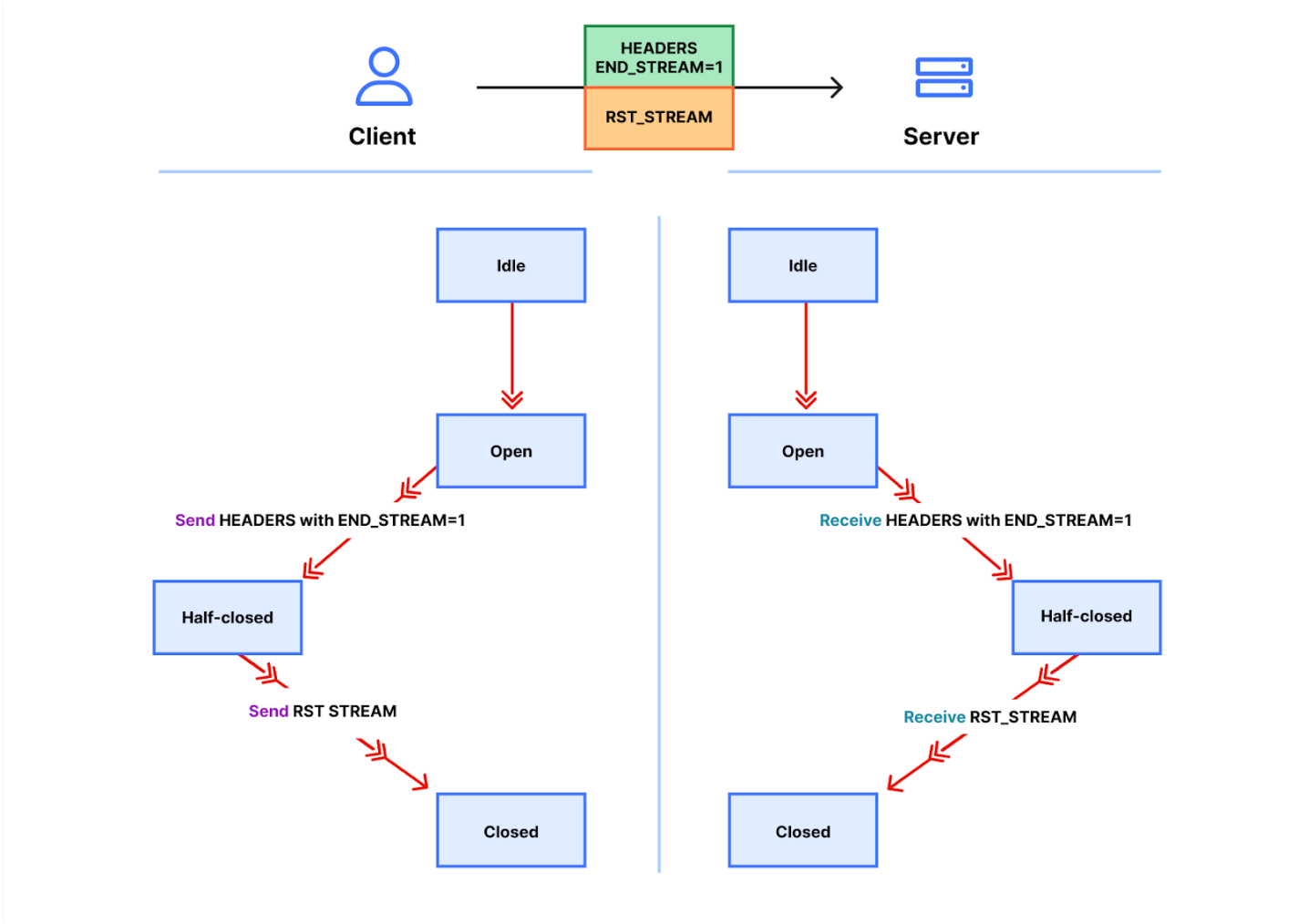
Earlier, we talked about client cancellation of in-flight requests. HTTP/2 supports this in a much more efficient way than HTTP/1.1. Rather than needing to tear down the whole connection, a client can send a RST_STREAM frame for a single stream. This instructs the server to stop processing the request and to abort the response, which frees up server resources and avoids wasting bandwidth.

Let's consider our previous example of 3 requests. This time the client cancels the request on stream 1 after all of the HEADERS have been sent. The server parses this RST_STREAM frame before it is ready to serve the response and instead only responds to stream 3 and 5:



Request cancellation is a useful feature. For example, when scrolling a webpage with multiple images, a web browser can cancel images that fall outside the viewport, meaning that images entering it can load faster. HTTP/2 makes this behaviour a lot more efficient compared to HTTP/1.1.

A request stream that is canceled, rapidly transitions through the stream lifecycle. The client's HEADERS with END_STREAM flag set to 1 transitions the state from **idle** to **open** to **half-closed**, then RST_STREAM immediately causes a transition from **half-closed** to **closed.**



Recall that only streams that are in the open or half-closed state contribute to the stream concurrency limit. When a client cancels a stream, it instantly gets the ability to open another stream in its place and can send another request immediately. This is the crux of what makes CVE-2023-44487 work.

Rapid resets leading to denial of service

HTTP/2 request cancellation can be abused to rapidly reset an unbounded number of streams. When an HTTP/2 server is able to process client-sent RST_STREAM frames and tear down state quickly enough, such rapid resets do not cause a problem. Where issues start to crop up is when there is any kind of delay or lag in tidying up. The client can churn through so many requests that a backlog of work accumulates, resulting in excess consumption of resources on the server.

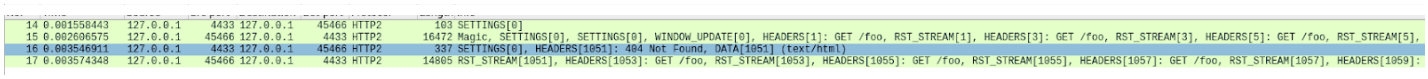
A common HTTP deployment architecture is to run an HTTP/2 proxy or load-balancer in front of other components. When a client request arrives it is quickly dispatched and the actual work is done as an asynchronous activity somewhere else. This allows the proxy to handle client traffic very efficiently. However, this separation of concerns can make it hard for the proxy to tidy up the in-process jobs. Therefore, these deployments are more likely to encounter issues from rapid resets.

When Cloudflare's reverse proxies process incoming HTTP/2 client traffic, they copy the data from the connection's socket into a buffer and process that buffered data in order. As each request is read (HEADERS and DATA frames) it is dispatched to an upstream service. When RST_STREAM frames are read, the local state for the request is torn down and the upstream is notified that the request has been canceled. Rinse and repeat until the entire buffer is consumed. However this logic can be abused: when a malicious client started sending an enormous chain of requests and resets at the start of a connection, our servers would eagerly read them all and create stress on the upstream servers to the point of being unable to process any new incoming request.

Something that is important to highlight is that stream concurrency on its own cannot mitigate rapid reset. The client can churn requests to create high request rates no matter the server's chosen value of SETTINGS_MAX_CONCURRENT_STREAMS.

Rapid Reset dissected

Here's an example of rapid reset reproduced using a proof-of-concept client attempting to make a total of 1000 requests. I've used an off-the-shelf server without any mitigations; listening on port 443 in a test environment. The traffic is dissected using Wireshark and filtered to show only HTTP/2 traffic for clarity. Download the pcap to follow along.



It's a bit difficult to see, because there are a lot of frames. We can get a quick summary via Wireshark's Statistics > HTTP2 tool:

Topic / Item	Cour
HTTP2	2008
Type	2008
HEADERS	1001
RST_STREAM	1000
SETTINGS	4
WINDOW_UPDATE	1
GOAWAY	1
DATA	1

The first frame in this trace, in packet 14, is the server's SETTINGS frame, which advertises a maximum stream concurrency of 100. In packet 15, the client sends a few control frames and then starts making requests that are rapidly reset. The first HEADERS frame is 26 bytes long, all subsequent HEADERS are only 9 bytes. This size difference is due to a compression technology called HPACK. In total, packet 15 contains 525 requests, going up to stream 1051.

```
▾ HyperText Transfer Protocol 2
  ▾ Stream: HEADERS, Stream ID: 1, Length 26, GET /foo
      Length: 26
      Type: HEADERS (1)
    ▸ Flags: 0x05, End Headers, End Stream
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
      Header Block Fragment: 048362539f87418a089d5c0b8170dc69a659827a852f91d35d05
      [Header Length: 112]
      [Header Count: 5]
    ▸ Header: :path: /foo
    ▸ Header: :scheme: https
    ▸ Header: :authority: 127.0.0.1:4433
    ▸ Header: :method: GET
    ▸ Header: user-agent: example
      [Full request URI: https://127.0.0.1:4433/foo]
▾ HyperText Transfer Protocol 2
  ▾ Stream: RST_STREAM, Stream ID: 1, Length 4
      Length: 4
      Type: RST_STREAM (3)
    ▸ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      Error: CANCEL (8)
▾ HyperText Transfer Protocol 2
  ▾ Stream: HEADERS, Stream ID: 3, Length 9, GET /foo
      Length: 9
      Type: HEADERS (1)
    ▸ Flags: 0x05, End Headers, End Stream
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0011 = Stream Identifier: 3
      [Pad Length: 0]
      Header Block Fragment: 048362539f87bf82be
      [Header Length: 112]
      [Header Count: 5]
    ▸ Header: :path: /foo
    ▸ Header: :scheme: https
    ▸ Header: :authority: 127.0.0.1:4433
    ▸ Header: :method: GET
    ▸ Header: user-agent: example
      [Full request URI: https://127.0.0.1:4433/foo]
```

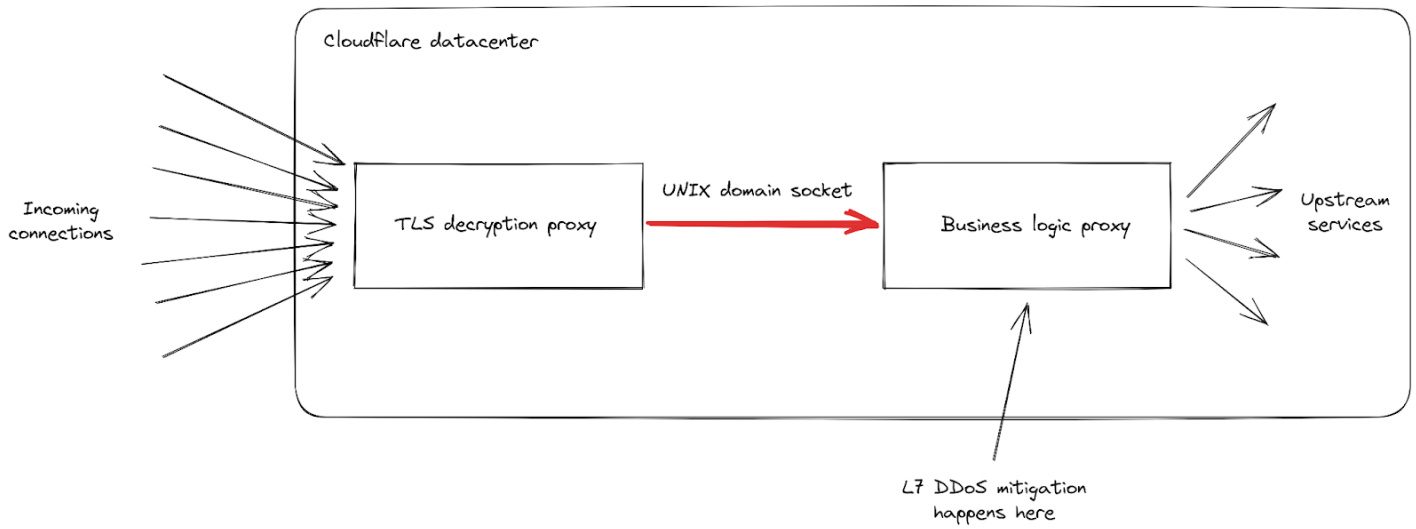
Interestingly, the RST_STREAM for stream 1051 doesn't fit in packet 15, so in packet 16 we see the server respond with a 404 response. Then in packet 17 the client does send the RST_STREAM, before moving on to sending the remaining 475 requests.

Note that although the server advertised 100 concurrent streams, both packets sent by the client sent a lot more HEADERS frames than that. The client did not have to wait for any return traffic from the server, it was only limited by the size of the packets it could send. No server RST_STREAM frames are seen in this trace, indicating that the server did not observe a concurrent stream violation.

Impact on customers

As mentioned above, as requests are canceled, upstream services are notified and can abort requests before wasting too many resources on it. This was the case with this attack, where most malicious requests were never forwarded to the origin servers. However, the sheer size of these attacks did cause some impact.

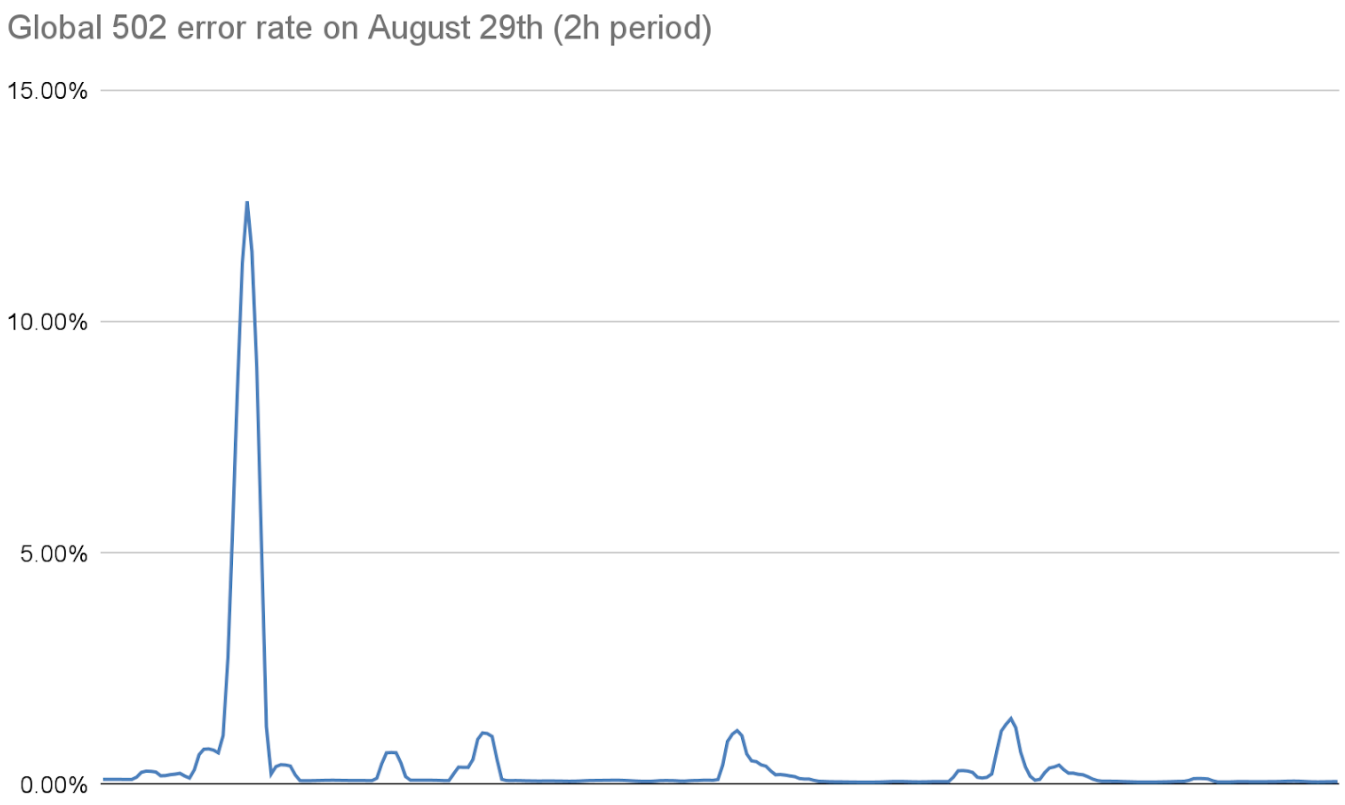
First, as the rate of incoming requests reached peaks never seen before, we had reports of increased levels of 502 errors seen by clients. This happened on our most impacted data centers as they were struggling to process all the requests. While our network is meant to deal with large attacks, this particular vulnerability exposed a weakness in our infrastructure. Let's dig a little deeper into the details, focusing on how incoming requests are handled when they hit one of our data centers:



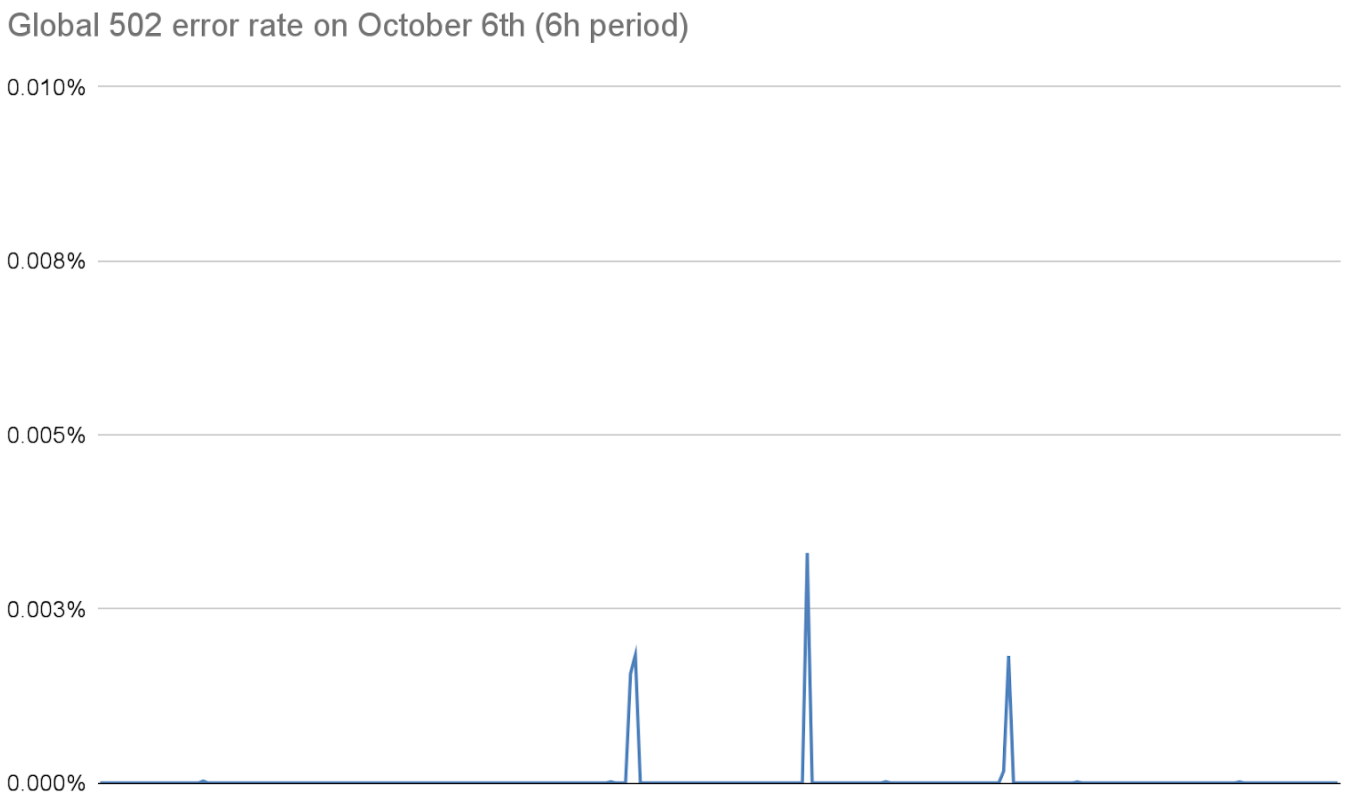
We can see that our infrastructure is composed of a chain of different proxy servers with different responsibilities. In particular, when a client connects to Cloudflare to send HTTPS traffic, it first hits our TLS decryption proxy: it decrypts TLS traffic, processes HTTP 1, 2 or 3 traffic, then forwards it to our "business logic" proxy. This one is responsible for loading all the settings for each customer, then routing the requests correctly to other upstream services — and more importantly in our case, **it is also responsible for security features**. This is where L7 attack mitigation is processed.

The problem with this attack vector is that it manages to send a lot of requests very quickly in every single connection. Each of them had to be forwarded to the business logic proxy before we had a chance to block it. As the request throughput became higher than our proxy capacity, the pipe connecting these two services reached its saturation level in some of our servers.

When this happens, the TLS proxy cannot connect anymore to its upstream proxy, this is why some clients saw a bare "502 Bad Gateway" error during the most serious attacks. It is important to note that, as of today, the logs used to create HTTP analytics are also emitted by our business logic proxy. The consequence of that is that these errors are not visible in the Cloudflare dashboard. Our internal dashboards show that about 1% of requests were impacted during the initial wave of attacks (before we implemented mitigations), with peaks at around 12% for a few seconds during the most serious one on August 29th. The following graph shows the ratio of these errors over a two hours while this was happening:



We worked to reduce this number dramatically in the following days, as detailed later on in this post. Both thanks to changes in our stack and to our mitigation that reduce the size of these attacks considerably, this number today is effectively zero.

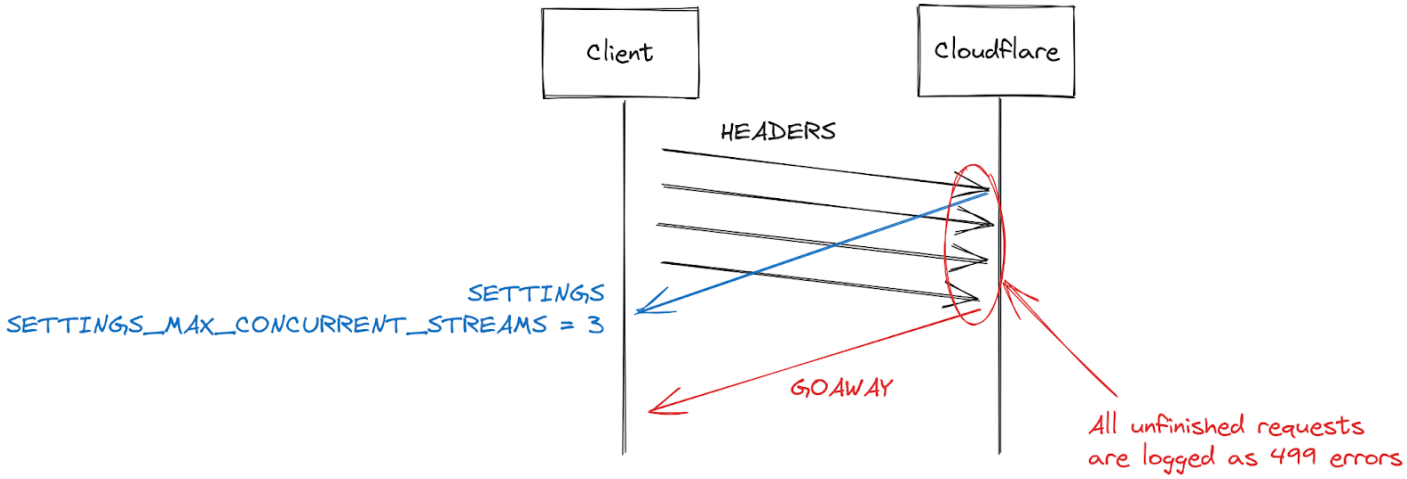


499 errors and the challenges for HTTP/2 stream concurrency

Another symptom reported by some customers is an increase in 499 errors. The reason for this is a bit different and is related to the maximum stream concurrency in a HTTP/2 connection detailed earlier in this post.

HTTP/2 settings are exchanged at the start of a connection using SETTINGS frames. In the absence of receiving an explicit parameter, default values apply. Once a client establishes an HTTP/2 connection, it can wait for a server's SETTINGS (slow) or it can assume the default values and start making requests (fast). For SETTINGS_MAX_CONCURRENT_STREAMS, the default is effectively

unlimited (stream IDs use a 31-bit number space, and requests use odd numbers, so the actual limit is 1073741824). The specification recommends that a server offer no fewer than 100 streams. Clients are generally biased towards speed, so don't tend to wait for server settings, which creates a bit of a race condition. Clients are taking a gamble on what limit the server might pick; if they pick wrong the request will be rejected and will have to be retried. Gambling on 1073741824 streams is a bit silly. Instead, a lot of clients decide to limit themselves to issuing 100 concurrent streams, with the hope that servers followed the specification recommendation. Where servers pick something below 100, this client gamble fails and streams are reset.



There are many reasons a server might reset a stream beyond concurrency limit overstepping. HTTP/2 is strict and requires a stream to be closed when there are parsing or logic errors. In 2019, Cloudflare developed several mitigations in response to HTTP/2 DoS vulnerabilities. Several of those vulnerabilities were caused by a client misbehaving, leading the server to reset a stream. A very effective strategy to clamp down on such clients is to count the number of server resets during a connection, and when that exceeds some threshold value, close the connection with a GOAWAY frame. Legitimate clients might make one or two mistakes in a connection and that is acceptable. A client that makes too many mistakes is probably either broken or malicious and closing the connection addresses both cases.

While responding to DoS attacks enabled by CVE-2023-44487, Cloudflare reduced maximum stream concurrency to 64. Before making this change, we were unaware that clients don't wait for SETTINGS and instead assume a concurrency of 100. Some web pages, such as an image gallery, do indeed cause a browser to send 100 requests immediately at the start of a connection. Unfortunately, the 36 streams above our limit all needed to be reset, which triggered our counting mitigations. This meant that we closed connections on legitimate clients, leading to a complete page load failure. As soon as we realized this interoperability issue, we changed the maximum stream concurrency to 100.

Actions from the Cloudflare side

In 2019 several DoS vulnerabilities were uncovered related to implementations of HTTP/2. Cloudflare developed and deployed a series of detections and mitigations in response. CVE-2023-44487 is a different manifestation of HTTP/2 vulnerability. However, to mitigate it we were able to extend the existing protections to monitor client-sent RST_STREAM frames and close connections when they are being used for abuse. Legitimate client uses for RST_STREAM are unaffected.

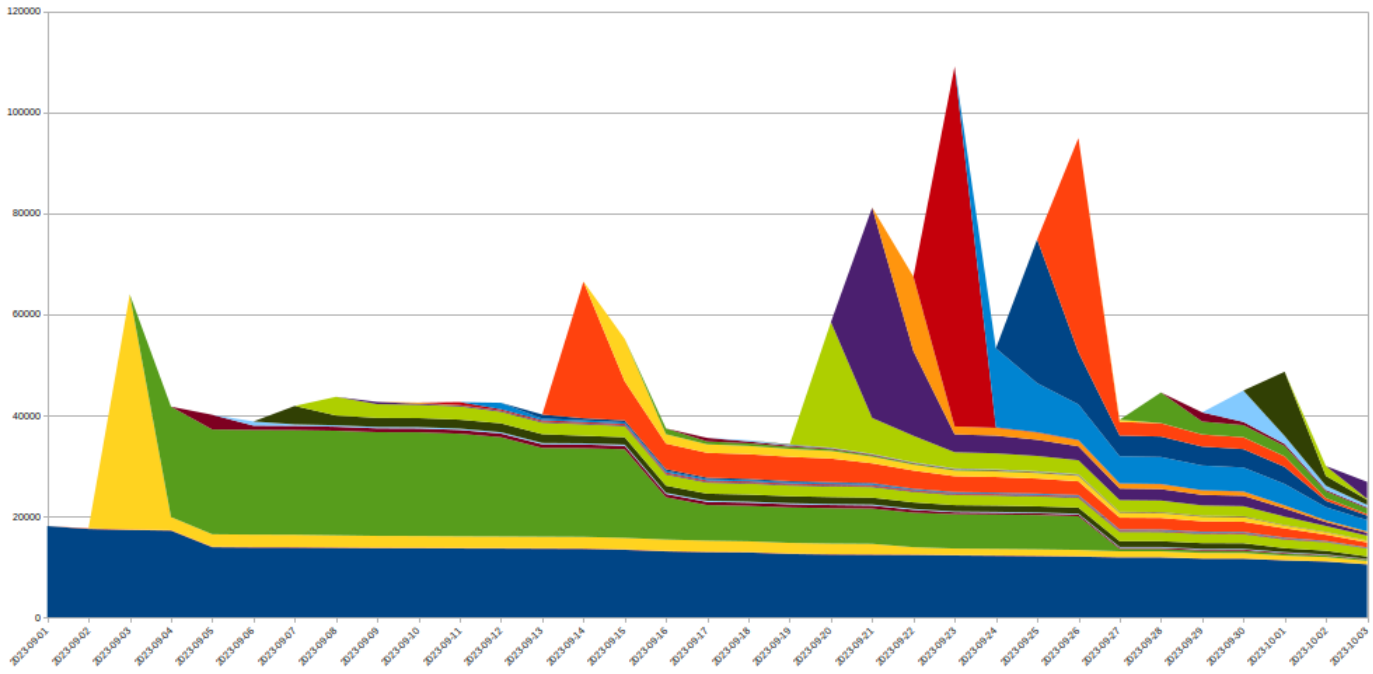
In addition to a direct fix, we have implemented several improvements to the server's HTTP/2 frame processing and request dispatch code. Furthermore, the business logic server has received improvements to queuing and scheduling that reduce unnecessary work and improve cancellation responsiveness. Together these lessen the impact of various potential abuse patterns as well as giving more room to the server to process requests before saturating.

Mitigate attacks earlier

Cloudflare already had systems in place to efficiently mitigate very large attacks with less expensive methods. One of them is named "IP Jail". For hyper volumetric attacks, this system collects the client IPs participating in the attack

and stops them from connecting to the attacked property, either at the IP level, or in our TLS proxy. This system however needs a few seconds to be fully effective; during these precious seconds, the origins are already protected but our infrastructure still needs to absorb all HTTP requests. As this new botnet has effectively no ramp-up period, we need to be able to neutralize attacks before they can become a problem.

To achieve this we expanded the IP Jail system to protect our entire infrastructure: once an IP is "jailed", not only it is blocked from connecting to the attacked property, we also forbid the corresponding IPs from using HTTP/2 to any other domain on Cloudflare for some time. As such protocol abuses are not possible using HTTP/1.x, this limits the attacker's ability to run large attacks, while any legitimate client sharing the same IP would only see a very small performance decrease during that time. IP based mitigations are a very blunt tool — this is why we have to be extremely careful when using them at that scale and seek to avoid false positives as much as possible. Moreover, the lifespan of a given IP in a botnet is usually short so any long term mitigation is likely to do more harm than good. The following graph shows the churn of IPs in the attacks we witnessed:



As we can see, many new IPs spotted on a given day disappear very quickly afterwards.

As all these actions happen in our TLS proxy at the beginning of our HTTPS pipeline, this saves considerable resources compared to our regular L7 mitigation system. This allowed us to weather these attacks much more smoothly and now the number of random 502 errors caused by these botnets is down to zero.

Observability improvements

Another front on which we are making change is observability. Returning errors to clients without being visible in customer analytics is unsatisfactory. Fortunately, a project has been underway to overhaul these systems since long before the recent attacks. It will eventually allow each service within our infrastructure to log its own data, instead of relying on our business logic proxy to consolidate and emit log data. This incident underscored the importance of this work, and we are redoubling our efforts.

We are also working on better connection-level logging, allowing us to spot such protocol abuses much more quickly to improve our DDoS mitigation capabilities.

Conclusion

While this was the latest record-breaking attack, we know it won't be the last. As attacks continue to become more sophisticated, Cloudflare works relentlessly to proactively identify new threats — deploying countermeasures to our global network so that our millions of customers are immediately and automatically protected.

Cloudflare has provided free, unmetered and unlimited DDoS protection to all of our customers since 2017. In addition, we offer a range of additional security

Reverse traceroute

Ethan Katz-Bassett[*] *Harsha V. Madhyastha*[†] *Vijay Kumar Adhikari*[‡] *Colin Scott*[*]
Justine Sherry[*] *Peter van Wesep*[*] *Thomas Anderson*[*] *Arvind Krishnamurthy*[*]

Abstract

Traceroute is the most widely used Internet diagnostic tool today. Network operators use it to help identify routing failures, poor performance, and router misconfigurations. Researchers use it to map the Internet, predict performance, geolocate routers, and classify the performance of ISPs. However, traceroute has a fundamental limitation that affects all these applications: it does not provide reverse path information. Although various public traceroute servers across the Internet provide some visibility, no general method exists for determining a reverse path from an arbitrary destination.

In this paper, we address this longstanding limitation by building a reverse traceroute system. Our system provides the same information as traceroute, but for the reverse path, and it works in the same case as traceroute, when the user may lack control of the destination. We use a variety of measurement techniques to incrementally piece together the path from the destination back to the source. We deploy our system on PlanetLab and compare reverse traceroute paths with traceroutes issued from the destinations. In the median case our tool finds 87% of the hops seen in a directly measured traceroute along the same path, versus only 38% if one simply assumes the path is symmetric, a common fallback given the lack of available tools. We then illustrate how we can use our reverse traceroute system to study previously unmeasurable aspects of the Internet: we present a case study of how a content provider could use our tool to troubleshoot poor path performance, we uncover more than a thousand peer-to-peer AS links invisible to current topology mapping efforts, and we measure the latency of individual backbone links with average error under a millisecond.

1 Introduction

Traceroute is a simple and widely used Internet diagnostic tool. It measures the sequence of routers from the source to the destination, supplemented by round-trip delays to each hop. Operators use it to investigate routing failures and performance problems [39]. Researchers use it as the basis for Internet maps [1, 22, 34], path prediction [22], geolocation [42, 14], ISP performance analysis [25], and anomaly detection [46, 19, 44, 43].

However, traceroute has a fundamental limitation – it provides no reverse path information, despite the fact that policy routing and traffic engineering mean that paths are generally asymmetric [15]. As Richard Steenbergen, CTO for nLayer Communications, put it at a recent tutorial for network operators on troubleshooting, "the number one go-to tool is traceroute," but "asymmetric paths [are] the number one plague of traceroute" because "the reverse path itself is completely invisible" [39].

This invisibility hinders operators. For instance, although Google has data centers distributed around the world, 20% of client prefixes experience unreasonably high latency, even with a nearby server. In working with a Google group trying to improve this performance, we found that we would have been able to more precisely troubleshoot problems if we could measure the path from clients back to Google [21].

Similarly, the lack of reverse path information restricts researchers. Traceroute's inability to measure reverse paths forces unrealistic assumptions of symmetry on systems with goals ranging from path prediction [22], geolocation [42, 14], ISP performance analysis [25], and prefix hijack detection [46]. Recent work shows that measured topologies miss many of the Internet's peer-to-peer links [29, 16] because mapping projects [1, 22, 34] lack the ability to measure paths from arbitrary destinations.

Faced with this shortcoming with the traceroute tool, operators and researchers turn to various limited workarounds. Surprisingly, network operators often resort to posting problems on operator mailing lists asking others to issue traceroutes to help diagnosis [30, 41]. Public web-accessible traceroute servers hosted at various locations around the world provide some help, but their numbers are limited. Without a server in every network, one cannot know whether any of those available have a path similar to the one of interest. Further, they are not intended for the heavy load incurred by regular monitoring. A few modern systems attempt to deploy traceroute clients on end-user systems around the world [34, 9], but none of them are close to allowing an arbitrary user to trigger an on-demand traceroute towards the user from anywhere in the world.

Our goal is to address this basic restriction of traceroute by building a tool to provide the same basic information as traceroute – IP-address-level hops along the path, plus round-trip delay to each – but along the reverse path from the destination back to the source. We have implemented our reverse traceroute system and make

[*]Dept. of Computer Science, Univ. of Washington, Seattle.
[†]Dept. of Computer Science, Univ. of California, San Diego.
[‡]Dept. of Computer Science, Univ. of Minnesota.

it available at http://revtr.cs.washington.edu. While traceroute runs as a stand-alone program issuing probes on its own behalf, ours is a distributed system comprised of a few tens to hundreds of vantage points, owing to the difficulty in measuring reverse paths. As with traceroute, our reverse traceroute tool does not require control of the destination, and hence can be used with arbitrary targets. All our tool requires of the target destination is an ability to respond to probes, the same requirement as standard traceroute. It does not require new functionality from routers or other network components.

Our system builds a reverse path incrementally, using a variety of methods to measure reverse hops, and stitching them together into a path. We combine the view of multiple vantage points to gather information unavailable from any single one. We start by measuring the paths from the vantage points to the source. This limited atlas of a few hundred or thousand routes to the source serves to bootstrap the rest of our measurements, allowing us to measure the path from an arbitrary destination by building back the path from the destination until it intersects the atlas. We use three main measurement techniques to build backwards. First, we rely on the fact that Internet routing is generally destination-based, allowing us to piece together the path one hop at a time. Second, we employ the IP timestamp and record route options to identify hops along the reverse path. Third, we use limited source spoofing – spoofing from one vantage point as another – to use the vantage point in the best position to make the measurement. This controlled spoofing allows us to overcome many of the limitations inherent in using IP options [36, 35, 13], while remaining safe, as the spoofed source address is one of our hosts. Just as many projects use traceroute, others have used record route and spoofing for other purposes. Researchers used record route to identify aliases and generate accurate topologies [35], and our earlier work used spoofing to characterize reachability problems [19]. In this work, we are the first to show that the combination of these techniques can be used to measure arbitrary reverse paths.

Experienced users realize that, while traceroute is useful, it has numerous limitations and caveats, and can be potentially misleading [39]. Similarly, our tool has limitations and caveats. Section 5.1 includes a thorough discussion of how the output of our tool might differ from a direct traceroute from the destination to the source, as well as how both might differ from the actual path traversed by traffic. Just as traceroute provides little visibility when routers do not send TTL-expired messages, our technique relies on routers honoring IP options. When our measurement techniques fail to discern a hop along the path, we fall back on assuming the hop is traversed symmetrically; our evaluation results show that, in the median (mean) case for paths between PlanetLab sites, we measure 95% (87%) of hops without assuming symmetry. The need to assume symmetry in cases of an unresponsive hop points to a limitation of our tool compared to traceroute; whereas traceroute can often measure past an unresponsive hop or towards an unreachable destination, our tool must sometimes guess that it is measuring the proper path.

We rely on routers to be "friendly" to our techniques, yet some of our techniques have the potential for abuse and can be tricky for novices to use without causing disturbances. As we ultimately want our tool widely used operationally, we have attempted to pursue our approach in a way that encourages continued router support. Our system performs measurements in a way that emphasizes network friendliness, controlling probe rate across all measurements. We presented the work early on at NANOG [28] and RIPE [32] conferences, and so far the response from operators has been positive towards supporting our methods (including source spoofing). We believe the goal of wide use is best served by a single, coordinated system that services requests from all users.

We evaluate the effectiveness of our system as deployed today, though it should improve as we add vantage points. We find that, in the median (mean) case for paths between PlanetLab sites, our technique reveals 87% (83%) of the routers and 100% (94%) of the points-of-presence (PoPs), compared to a traceroute issued from the destination. Paths between public traceroute servers and PlanetLab show similar results. Because our technique requires software at the source, our evaluation is limited to paths back to PlanetLab nodes we control. We believe our reverse traceroute system can be useful in a range of contexts, and we provide three illustrative examples. We present a case study of how a content provider could use our tool to troubleshoot poor reverse path performance. We also uncover thousands of links at core Internet exchange points that are invisible to current topology mapping efforts. We use our reverse traceroute tool to measure link latencies in the Sprint backbone network with less than a millisecond of error, on average.

2 Background

In this section, we provide the reader some background on Internet routing and traceroute.

Internet routing: First, a router generally determines the route on which to forward traffic based only on the destination. With a few caveats such as load-balancing and tunneling, the route from a given point towards a particular destination is consistent for all traffic, regardless of its source. While certain tunnels may violate this assumption, best practices encourage tunnels that appear as atomic links. Second, asymmetry between forward and reverse paths stems from multiple causes. An AS is free to choose its next hop among the alternatives, whether or

not that leads to a symmetric route. Two adjacent ASes may use different peering points in the two directions due to policies such as early-exit/hot-potato routing. Even within an individual AS, traffic engineering objectives may lead to different paths.

Standard traceroute tool: Traceroute measures the sequence of routers from a source to a destination, without requiring control of the destination. When traceroute was originally developed, most paths were symmetric, but that assumption no longer holds. Traceroute works by sending a series of packets to the destination, each time incrementing the time-to-live (TTL) from an initial value of one, in order to get ICMP TTL exceeded responses from each router on the path in turn. Each response will have an IP address of an interface of the corresponding router. Additionally, traceroute measures the time from the sending of each packet to the receipt of the response, yielding a round-trip latency to each intermediate router. Because the destination resets the TTL in its reply, traceroute only works in the forward direction.

The path returned by traceroute is a feasible, but possibly inaccurate route. First, each hop comes from a response to a different probe packet, and the different probes may take different paths for reasons including contemporaneous routing changes or load balancing. The Paris traceroute customizes probe packets to provide consistent results across flow-based load balancers, as well as to systematically explore the load-balancing options [2]. For all our traceroutes, we use the Paris option that measures a single consistent path. Second, some routers on the path may not respond. For example, some routers may be configured to rate-limit responses or to not respond at all. Third, probe traffic may be treated differently than data traffic.

Despite these caveats, traceroute has proved to be extremely useful. Essential to traceroute's utility is its universality, in that it does not require anything of the destination other than an ability to respond to probe packets.

3 Reverse Traceroute

We seek to build a reverse path tool equivalent to traceroute. Like traceroute, ours should work universally without requiring control of a destination, and it should use only features available in the Internet as it exists today. The reverse traceroute tool should return IP addresses of routers along the reverse path from a destination back to the source, as well as the round-trip delay from those routers to the source.

At a high level, the source requests a path from our system, which coordinates probes from the source and from a set of distributed vantage points to discover the path. First, distributed vantage points issue traceroutes to the source, yielding an atlas of paths towards it (Fig. 1(a)). This atlas provides a rich, but limited in scope, view of how parts of the Internet route towards the source. We use this limited view to bootstrap measurement of the desired path. Because Internet routing is generally destination-based, we assume that the path to the source from any hop in the atlas is fixed (over short time periods) regardless of how any particular packet reaches that hop; once the path from the destination to the source reaches a hop in the atlas, we use the atlas to derive the remainder of the path. Second, using techniques we explain in Sections 3.1 and 3.2, we measure the path back from the destination incrementally until it intersects this atlas (Fig. 1(b)). Finally, as shown in an example in Section 3.3, we merge the two components of the path, the destination-specific part measured from the destination until it intersects the atlas, and the atlas-derived path from this intersection back to the source, to yield a complete path (Fig. 1(c)).

3.1 Identify Reverse Hops with IP Options

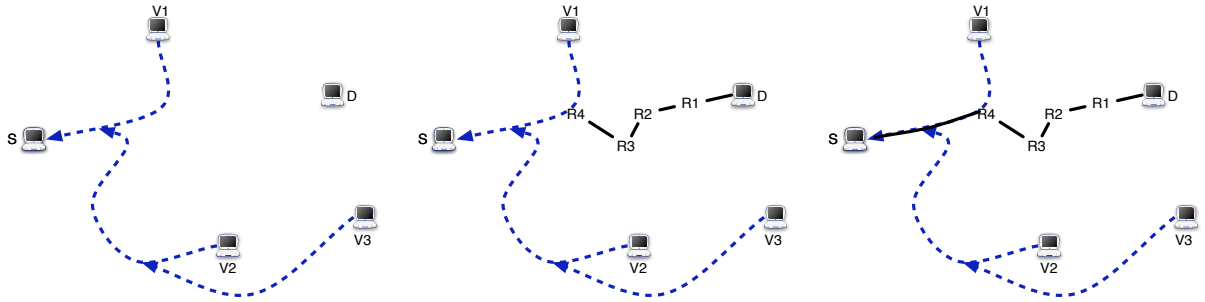
We use two basic measurement primitives, the Record Route and Timestamp IP options. While TTL values are reset by the destination, restricting traceroute to measuring only on the forward path, IP options are generally reflected in the reply from a destination, so routers along both the forward and reverse path process them. We briefly explain how the options work:

IP Record-route option (*RR*): With this option set, a probe records the router interfaces it encounters. The IP standard limits the number of recorded interfaces to 9; once those fill, no more interfaces are recorded.

IP timestamp option (*TS*): IP allows probes to query a set of specific routers for timestamps. Each probe can specify up to four IP addresses, in order; if the probe traverses the router matching the next IP address that has yet to be stamped, the router records a timestamp. The addresses are ordered, so a router will not timestamp if its IP address is in the list but is not the next one.

We use these options to gather reverse hops as follows:

- *RR-Ping*$(S \rightarrow D)$: As shown in Figure 2(a)), the source S issues an ICMP Echo Request (henceforth *ping*) probe to D with the RR option enabled. If RR slots remain when the destination sends its response, then routers on the reverse path will record some of that route. This allows a limited measurement of the reverse path, as long as the destination is fewer than 9 hops from the source.
- *TS-Query-Ping*$(S \rightarrow D|D, R)$: As shown in Figure 2(b)), the source S issues an ICMP ping probe to D with the timestamp query enabled for the ordered pair of IP addresses D and R. R will record its timestamp only if it is encountered by the probe after D has stamped the packet. In other words, if S receives a timestamp for R, then it knows R appears

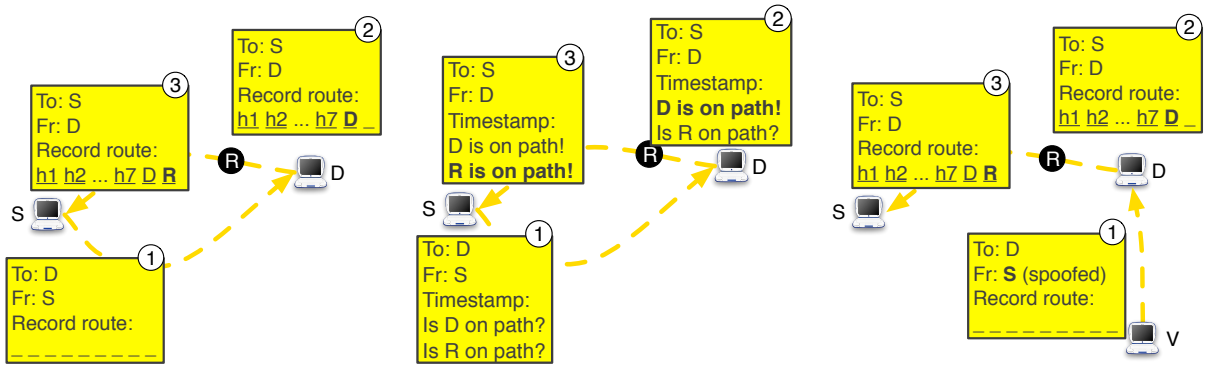


(a) Vantage points traceroute to S, creating an atlas of known paths.

(b) Vantage points measure path from D until it intersects a path in the atlas.

(c) Combine to yield complete path.

Figure 1: High-level overview of the reverse traceroute technique. We explain how to measure from D back to the atlas in §3.1-3.2.



(a) S sends a record-route ping. The header includes slots for 9 IP addresses to be recorded (1). If the packet reaches D with slots remaining, D adds itself (2), and routers on the reverse path fill the remaining slots (3).

(b) S sends a timestamp ping, asking first for D to provide a stamp if encountered, then for R to provide one (1). If D supports the timestamp option, it fills out a timestamp (2). Because the timestamp requests are ordered, R only fills out a timestamp if encountered after D, necessarily on the reverse path (3).

(c) Vantage point V sends a record-route ping to D, spoofing as S (1). D replies to S (2), allowing S to discover that R is on the reverse path (3). We use this technique when S is out of record-route range of D, but V is close enough.

Figure 2: Three measurement techniques that allow us to establish that R is on the reverse path from D back to S. In §4, we give two techniques, akin to (c), that use spoofing to overcome limitations in timestamp support.

on the reverse path. For our purposes, the value of the timestamp is meaningless; we just care whether or not a particular router processes the packet. Thus, if we guess a router on the return path, the TS option can confirm our hypothesis.

We use existing network topology information – specifically IP-level connectivity of routers from Internet mapping efforts [22] – to determine candidate sets of routers for the reverse path. Routers adjacent to D in the topology are potential next hops; we use timestamp query probes to check whether any of these potential next hops is on the path from D to S.

Note that there are some caveats to using the probes outlined above. One is that from each vantage point, only a fraction of routers will be reachable within record route's limit of 9 hops. Another is that some ISPs filter and drop probes with IP options set. Further, some routers do not process the IP options in the prescribed manner. Fortunately, we can overcome these limitations in the common case by carefully orchestrating the measurements from a diverse set of vantage points.

3.2 Spoof to Best Use Record Route

A source-spoofed probe (henceforth referred to as a spoofed probe) is one in which the prober sets the source address in the packet to one other than its own. We use a limited form of spoofing, where we replace the source address in a probe with the "true" source of the reverse traceroute. This form of spoofing is an extremely powerful measurement tool. When V probes D spoofing as S, D's response will go to S; we refer to V as the spoofer and S as the receiver. This method allows the probe to traverse the path from D to S without having to traverse the path from S to D and without having a vantage point in D's prefix. We could hypothetically achieve a similar probe trajectory using loose source routing (from V

to S, but source routed through D) [31]. However, a source-routed packet can be identified and filtered anywhere along the path, and such packets are widely filtered [3], too often to be useful in our application. On the other hand, if a spoofed packet is not ingress filtered near the spoofer, it thereafter appears as a normal packet; we can use a source capable of spoofing to probe along any path. Based on our measurements to all routable prefixes, many routers that filter packets with the source route option do not filter packets with the timestamp or record route options. This difference is likely because source routing can be used to violate routing policy, whereas the timestamp and record route options cannot.

This arrangement allows us to use the most advantageous vantage point with respect to the particular measurement we want to perform. Our earlier system Hubble used limited spoofing to check one-way reachability [19]; we use it here to overcome limitations of IP options. Without spoofing, RR's 9 IP address limit restricts it to being useful only when S is near the target. However, as shown in Figure 2(c), if some vantage point V is close enough to reach the target within 8 RR slots, then we can probe from V spoofing as S to receive IP addresses on the path back to S. Similarly, spoofing can bypass problematic ASes and machines, such as those that filter timestamp-enabled packets or those that do not correctly implement the option.

Although spoofing is often associated with malicious intent, we use it in a very controlled, safe fashion. A node requests a reverse path measurement, then receives responses to probes sent by vantage points spoofing as it. No harm can come from causing one of our own nodes to receive measurement packets. This form of spoofing shares a purpose with the address rewriting done by middleboxes such as NATs, controlling the flow of traffic to a cooperative machine, rather than with malicious spoofing which seeks concealment. Since some ISPs filter spoofed packets, we test from each host and only send further spoofed probes where allowed. We have been issuing spoofed probes for over two years without complaint.

Roughly 20% of PlanetLab sites allow spoofing; this ability is not limited to PlanetLab: the Spoofer project tested 12,000 clients and found that 31% could send spoof packets [5]. Even if filtering increases, we believe, based on positive feedback from operators, that the value of our service will encourage an allowance (supported by router ACLs) for a small number of measurement nodes to issue spoofed probes using a restricted set of ports. An even simpler approach is to have routers rate limit these spoofed options packets (just as with UDP probes) and filter spoofed probes sent to broadcast addresses, thereby reducing the security concerns without diminishing their utility for network measurements.

3.3 Incrementally Build Paths

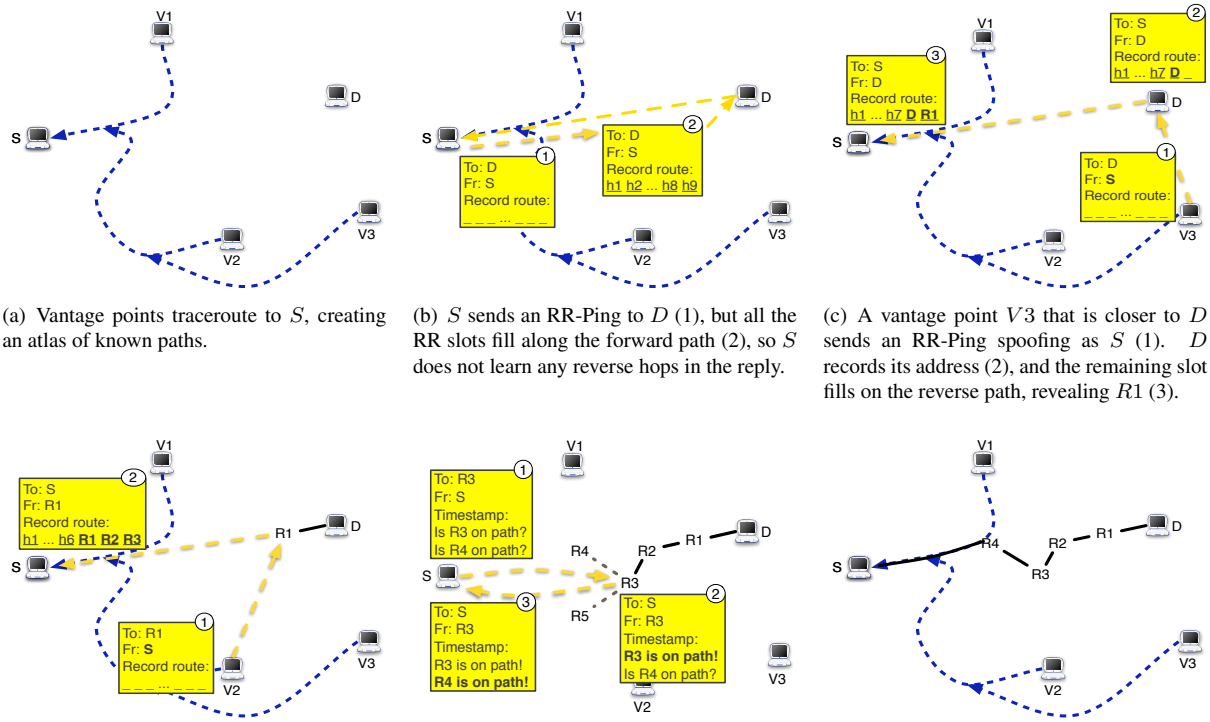
IP option-enabled probes, coupled with spoofing as S from another vantage point, give us the ability to measure a reverse hop from D on the path back to S. We can use the same techniques to stitch together a path incrementally – once we know the path from D goes through R, we need only determine the route at R towards S when attempting to discover the next hop. Because Internet routing is generally based on the destination, each intermediate router R we find on the path can become the new destination for a reverse traceroute back to the source. Further, if R is on a path from some vantage point V to S, then we can infer the rest of D's return path from R onward as being the same as V's. This assumption holds even in cases of packet-, flow-, and destination-based load balancing, so long as R balances traffic independently of other routers and of the source.

Figure 3 illustrates how we can compose the above set of techniques to determine the reverse path from D to S, when we have control over S and a set of other vantage points (V_1, V_2, V_3). We assume that we have a partial map of router-level connectivity, e.g., from a traditional offline mapping effort.

We begin by having the vantage points issue traceroute probes to S (Figures 1(a) and 3(a)). These serve as a baseline set of observed routes towards S that can be used to complete a partially inferred reverse path. We then issue *RR-Ping*$(S \rightarrow D)$ to determine if the source S is within 8 RR hops of the destination, i.e., whether a ping probe from S can reach D without filling up its entire quota of 9 RR hops (Figure 3(b))[1]. If the source is within 8 RR hops of D, this probe would determine at least the first hop on the reverse path, with further hops recovered in an iterative manner.

If the source is not within 8 hops, we determine whether some vantage point is within 8 RR hops of D (Section 4.5 describes how we do this). Let V_3 be one such vantage point. We then issue a spoofed RR ping probe from V_3 to D with the source address set to S (Figure 3(c)). This probe traverses the path $V_3 \rightarrow D \rightarrow S$ and records IP addresses encountered. The probe reveals R_1 to be on the reverse path from D to S. We then iterate over this process, with the newly found reverse hop as the target of our probes. For instance, we next determine a vantage point that is within 8 RR hops of R_1, which could be a different vantage point V_2. We use this new vantage point to issue a spoofed RR ping probe to determine the next hop on the reverse path (Figure 3(d)).

[1] This is not quite as simple as sending a TTL=8 limited probe, because of issues with record route implementations [35]. Some routers on the forward path might not record their addresses, thereby freeing up more slots for the reverse path, while some other routers might record multiple addresses or might record their address but not decrement or respond to TTL-limited probes.



(a) Vantage points traceroute to S, creating an atlas of known paths.

(b) S sends an RR-Ping to D (1), but all the RR slots fill along the forward path (2), so S does not learn any reverse hops in the reply.

(c) A vantage point $V3$ that is closer to D sends an RR-Ping spoofing as S (1). D records its address (2), and the remaining slot fills on the reverse path, revealing $R1$ (3).

(d) A vantage point $V2$ close to $R1$ sends an RR-Ping spoofing as S (1), discovering $R2$ and $R3$ on the reverse path (2).

(e) We use an Internet map to find routers adjacent to $R3$, then send each a TS-Query-Ping to verify which is on the reverse path.

(f) When we intersect a known path in our traceroute atlas, we assume the rest of the path from D follows that route.

Figure 3: Illustration of the incremental construction of a reverse path using diverse information sources.

In some cases, a single RR ping probe may determine multiple hops, as in the illustration with R_2 and R_3.

Now, consider the case where neither S nor any of the vantage points is within 8 hops of R_3. In that case, we consider the potential next hops to be routers adjacent to R_3 in the known topology. We issue timestamp probes to verify whether the next hop candidates R_4 and R_5 respond to timestamp queries *TS-Query-Ping*$(S \rightarrow D|D, R_4)$ and *TS-Query-Ping*$(S \rightarrow D|D, R_5)$ (as shown in Figure 3(e)). When R_4 responds, we know that it is adjacent to R_3 in the network topology and is on the reverse path from R_3, and so we assume it is the next hop on the reverse path. We continue to perform incremental reverse hop discovery until we intersect with a known path from a vantage point to S [2], at which point we consider that to be the rest of the reverse path (Figures 1(c) and 3(f)). Once the procedure has determined the hops in the reverse path, we issue pings from the source to each hop in order to determine the round-trip latencies.

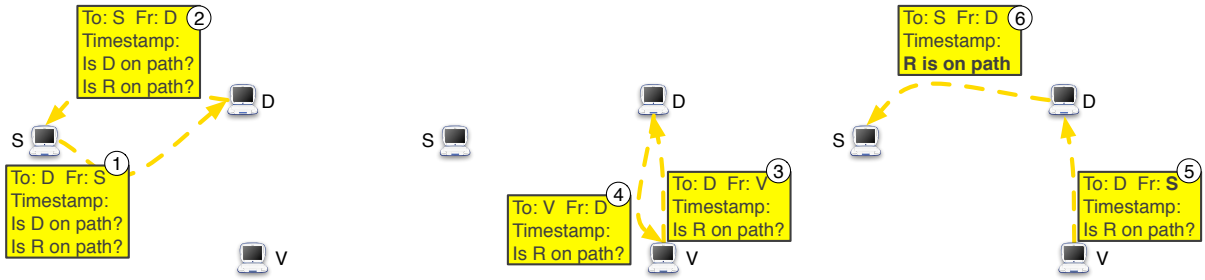
Sometimes, we may be unable to measure a reverse hop using any of our techniques, but we still want to provide the user with useful information. When reverse traceroute is unable to calculate the next hop in a path, the source issues a standard traceroute to the last known hop on the path. We then assume the last link is traversed symmetrically, and we try to calculate the rest of the reverse path from there. In Section 5.1 and Section 5.2, we present results that show that we usually do not have to assume many symmetric hops and that, even with this approximation, we still achieve highly accurate paths.

4 System Implementation

Section 3 describes how our techniques in theory would allow us to measure a reverse path. In this section, we discuss how we had to vary from that ideal description in response to realities of available vantage points and of router implementations. In addition, the following goals drive our system design:

- *Accuracy:* It should be robust to variations in how options-enabled packets are handled by routers.
- *Coverage:* The system should work for arbitrary destinations irrespective of ISP-specific configurations.
- *Scalability:* It needs to be selective with the use of vantage points and introduce as little measurement traffic as possible.

[2]Measurement techniques may discover different addresses on a router [36], so we determine intersections using alias data from topology mapping projects [20, 22, 35] and a state-of-the-art technique [4].



(a) S sends a timestamp ping to D (1) and receives a reply, but D has not filled out a timestamp (2).

(b) We find a V s.t., when V pings D asking for R's timestamp (3), it does not receive a stamp (4). This response indicates that R is not on V's path to D.

(c) V spoofs as S, pinging D (5), and S receives a timestamp for R (6). Because we established that R is not on V's path to D, R must be on the reverse path from D to S.

Figure 4: Example of how we discover a reverse hop with timestamp even though D does not stamp, as long as it replies.

4.1 Architecture

Our system consists of vantage points (VPs), which issue measurements, a controller, which coordinates the VPs to measure reverse paths, and sources, which request paths to them. We use a local machine at UW as a controller. When a VP starts up, it registers with the controller, which can then send it probe requests. A source runs our software to issue standard traceroutes, *RR-Pings*, and *TS-Query-Pings*, and to receive responses to probes from VPs spoofing as the source. However, it need not spoof packets itself. Currently, our source software only runs on Linux and (like the ICMP traceroute option `traceroute -I`) requires root permission. The controller receives requests from sources and combines the measurements from VPs to report reverse path information. While measuring a reverse traceroute, the controller queues up all incoming requests. When the ongoing measurement completes, the controller serves all requests in the queue as a batch, in synchronized rounds of probes. This design lets us carefully control the rate at which we probe any particular router, as well as to reuse measurements when a particular source requests multiple destinations.

We use topology maps from iPlane [22][3] to identify adjacent routers to test with *TS-Query-Pings* (Fig. 3(e)). To increase the set of possible next-hop routers, we consider the topology to be the union of maps from the previous 2 weeks. Since we verify the reverse hops using option-enabled pings, stale topology information makes our system less efficient but does not introduce error.

4.2 Current Deployment

Our current deployment uses one host at each of the more than 200 active PlanetLab sites as VPs to build an atlas of traceroutes to a source (Fig. 3(a)). Over the course of our study, 60+ PlanetLab sites allowed spoofed probes at least some of the time. We employ one host at each of these sites as spoofing nodes. Routers upstream from the other PlanetLab sites filter spoofed probes, so we did not spoof from them. We also use one host at each of 14 Measurement Lab sites [26], most of which allow spoofing. Various organizations provide public web-accessible traceroute servers, and we employ 1200 of them [3]. These nodes issue only traceroutes and cannot set IP options, spoof, or receive spoofed probes. We use them to expand the sets of known paths to our sources.

We have currently tested our client software only on PlanetLab (Linux) machines. We make it available as a demo; our website `http://revtr.cs.washington.edu` allows users to enter an IP address and measure the reverse path back from it to a PlanetLab node. Packaging the code for widespread deployment is future work. Because we have dozens of operators asking to use the system, we are being patient to avoid a launch that does not perform up to expectations.

4.3 Correcting for Variations in IP Options Support

We next explain how we compensate for variations in support for the timestamp option. When checking if R is on the reverse path from D, we normally ask for both D's and R's timestamp, to force R to only stamp on the reverse path. However, we found that, of the addresses in a day's iPlane atlas that respond to ping, 16.6% of the addresses respond to timestamp-enabled pings, but do not stamp, so we cannot use that technique to know that R stamped on the reverse path. Figure 4 illustrates how we use spoofing to address this behavior. Essentially, we find a VP V which we can establish does not have R on its path to D, then V pings D spoofing as S, asking for R's timestamp (but not D's). If S receives a stamp for R, it proves R is on the reverse path from D. This technique will not work if all vantage points have R on their paths. We examined iPlane traceroutes to destinations in 140,000 prefixes and found at least two adjacent hops for 55% of destinations.

[3] iPlane issues forward traceroutes from PlanetLab sites and traceroute servers to around 140K prefixes.

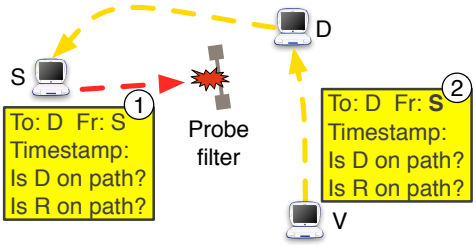


Figure 5: If a timestamp probe from S encounters a filter (1), we can often bypass it by spoofing as S from a different vantage point (2), as long as the filter is just on the forward path.

4.4 Avoiding Probe Filters to Improve Coverage

We next discuss techniques to improve the coverage of our measurements. Some networks may filter ICMP packets, and others filter packets with options enabled. In the course of measuring a reverse path, if a source attempts a TS or RR measurement and does not receive a response, we retry the measurement with a VP spoofing as the source. As seen in Figure 5, if filtering occurs only along the source's forward path, and the VP's forward path does not have a filter, the original source should receive the response.

We demonstrate the effectiveness of this approach on a small sample of 1000 IP addresses selected at random out of those in the iPlane topology known to respond to timestamp probes. The 1000 destinations include addresses in 662 ASes. We chose 10 spoofing PlanetLab vantage points we found to receive (non-spoofed) timestamp responses from the highest number of destinations, plus one host at each of the 209 working non-spoofing PlanetLab site. First, each non-spoofing node sent a series of timestamp pings to each destination; redundant probes account for loss due to something other than permanent filters. Of the 209 hosts, 103 received responses from at least 700 destinations; we dropped them from the experiment, as they do not experience significant filtering. Then, each spoofing vantage point sent 106 timestamp pings to each destination, spoofing as each of the remaining PlanetLab hosts in turn. Of these, 63 failed to receive any responses to either spoofed or non-spoofed probes; they are completely stuck behind filters or were not working. For the remaining 43 hosts, Figure 6 shows how many destinations each host receives responses from, both without and with spoofing. Our results show that some sites benefit significantly. In reverse traceroute's timestamp measurements, whenever the source does not receive a response, we retry with 5 spoofers. Since some vantage points have filter-free paths to most destinations, we use the 5 best overall, rather than choosing per destination. For the nodes that experience widespread filtering, spoofing enables a significant portion to still use timestamps as part of reverse traceroute. As we show in Section 5.2, our timestamp techniques help the overall coverage of the tool.

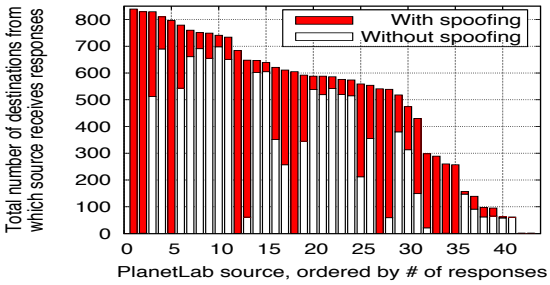


Figure 6: For 43 PlanetLab nodes, the number of destinations (out of 1000) from which the node receives timestamp responses. The graph shows the total number of unique destinations when sending the ping directly and then when also using 10 spoofers. The nodes are ordered by the total number of responding destinations. Other PlanetLab sites were tested but are not included in the graph: 103 did not experience significant filtering and 63 did not receive responses even with spoofing.

4.5 Selective Use of Vantage Points for Scalability

With spoofed RR, only nearby spoofers can find reverse hops, since each packet includes only 9 slots. Because many routers rate limit after only a few probes, we cannot send from many vantage points at once, in the hopes that one will prove close enough – the router might drop the probes from the VPs within range. Our goal is to determine which VPs are likely to be near a router before we probe it. Because Internet routing is generally based on the destination's prefix, a VP close to one address in a prefix is likely close to other addresses in the prefix.

Each day, we harvest the set of router IP addresses seen in the Internet atlas gathered by iPlane on the previous day and supplement the set with a recent list of pingable addresses [17]. Each day, every VP issues a record route ping to every address in the set. For each address, we determine the set of VPs that were near enough to discover reverse hops. We use this information in two ways during a reverse traceroute. First, if we encounter one of the probed addresses, we know the nearest VP to use. Second, if we encounter a new address, the offline probes provide a hint: the group of VPs within range of some address in the same prefix. Selecting the minimal number of vantage points to use from this group is an instance of the well known set cover optimization problem. We use the standard greedy algorithm to decide which VPs to use for a prefix, ordered by the number of additional addresses they cover within the prefix.

For a representative day, Figure 7 shows the coverage we achieve at given numbers of VPs per prefix. Our system determines the covering VPs for all prefixes, but the graph only includes prefixes for which we probed at least 15 addresses, as it is trivial to cover small prefixes. We see that, for most prefixes, we only need a small number of VPs. For example, in the median case, a single VP suffices for over 95% of addresses in the prefix, and we rarely need more than 4 VPs to cover the entire prefix.

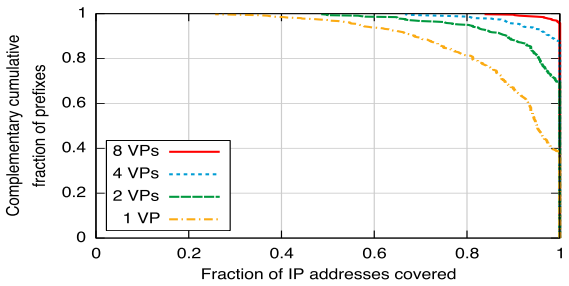


Figure 7: For prefixes in which iPlane observed $\geq$ 15 addresses, the fraction of the addresses for which we can find reverse hops using RR probes from a given number of vantage points per prefix. Note that we only include addresses within range of at least one vantage point. Prefixes with few addresses are trivial to cover using a small number of vantage points, so the graph excludes them to clearly show that we still only need a small number for most prefixes.

5 Evaluation

To test how well our reverse traceroute system can determine reverse paths, we consider evaluation settings that allow us to compare a reverse traceroute from D to S to a direct traceroute from D to S. A complete evaluation of the accuracy of our technique would require ground truth information about the path back from the destination. Obviously, we lack ground truth for the Internet, but we use two datasets, one PlanetLab-based and one using public traceroute servers, in which we can compare to a traceroute from D. For the reverse traceroute, we assume we do not control D and must measure the path using the techniques described in this paper. For the direct traceroute, we do control D and can simply issue a standard traceroute from D to S.

In the PlanetLab set, we employ as sources a host at each of 11 PlanetLab sites chosen at random from the spoofing nodes. As destinations, we use one host at each of the 200 non-spoofing PlanetLab sites that were working. Although such a set is not representative of the entire Internet, the destinations includes hosts in 35 countries. The measured reverse paths traversed 13 of the 14 transit-free commercial ISPs. Previous work observed route load balancing in many such networks [2], providing a good test for our techniques.

In the traceroute server set, we employ as sources a host at 10 of the same PlanetLab sites (one had gone down in the meantime). The 1200 traceroute servers we utilize belong to 186 different networks (many of which offer multiple traceroute servers with different locations). For each source, we choose a traceroute server at random from each of the 186 networks. We then issue a traceroute from the server to the PlanetLab source. Because in many cases we do not know the IP address of the traceroute server, we use the first hop along its path as the destination in our reverse traceroute measurements. When measuring a reverse traceroute from this destination back to the source, we exclude from our system all traceroute servers in the same network, to avoid providing our system with such similar paths as to make its task trivial.

5.1 Accuracy

How similar are the hops on a reverse traceroute to a direct traceroute from the destination back to the source? For the PlanetLab dataset, the *RevTR* line in Figure 8 depicts the fraction of hops seen on the direct traceroute that are also seen by reverse traceroute. Figure 9 shows the same for the traceroute server dataset. Note that, outside of this experimental setting, we would not normally have access to the direct traceroute from the destination. Using alias data from topology mapping projects [20, 22, 35] and aliases we discover using a state-of-the-art technique [4], we consider a traceroute hop and a reverse traceroute hop to be the same if they are aliases for the same router. We need to use alias information because the techniques may find different IP addresses on the same router [36]. For example, traceroute generally finds the ingress interface, whereas record route often returns the egress or loopback address. However, alias resolution is an active and challenging research area, and faulty aliases in the data we employ could lead us to falsely label two hops as equivalent. Conversely, missing aliases could cause us to label as different two interfaces on the same router. Because the alias sets we use are based on measurements from PlanetLab or similar vantage points, we likely have more complete alias data for our PlanetLab dataset than for our traceroute server dataset.

Using the available alias data, we find that the paths measured by our technique are quite similar to those seen by traceroute. In the median (mean) case, we measure 87% (83%) of the hops in the traceroute for the PlanetLab dataset. For the traceroute server dataset, we measure 75% (74%) of the hops in the direct traceroute, but 28% (29%) of the the hops discovered by reverse traceroute do not appear in the corresponding traceroute.

The figures also compare reverse traceroute to other potential ways of estimating the reverse path. All techniques used the same alias resolution. Researchers often (sometimes implicitly) assume symmetry, and operators likewise rely on forward path measurements when they need reverse ones. The *Guess Fwd* lines depict how many of the hops seen in a traceroute from R to S are also seen in a traceroute from S to R. In the median (mean) case, the forward path shares 38% (39%) of the reverse path's hops for the PlanetLab dataset and 40% (43%) for the traceroute server dataset. Another approach would be to measure traceroutes from a set of vantage points to the source. Using iPlane's PlanetLab

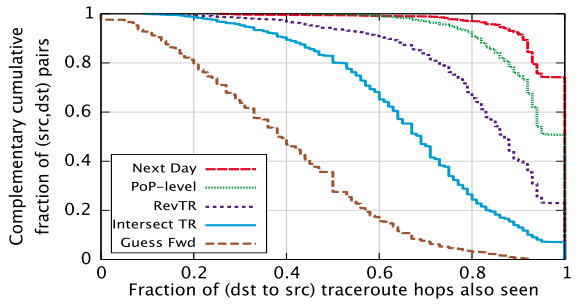


Figure 8: For reverse traceroute and techniques for approximating the reverse path, the fraction of hops on a direct traceroute from the destination to the source that the technique also discovers. Uses our PlanetLab dataset (reverse paths from 200 PlanetLab destinations back to 11 PlanetLab sources). [Key labels are in the same top-to-bottom order as the lines.]

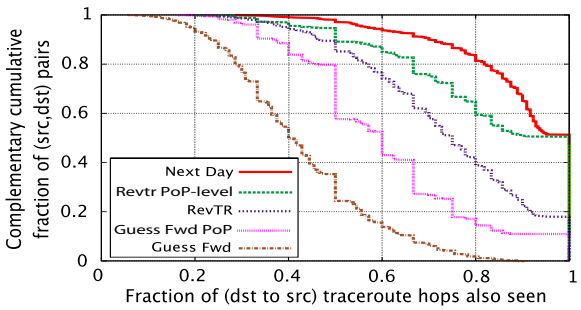


Figure 9: For reverse traceroute and techniques for approximating the reverse path, the fraction of hops on a direct traceroute from the destination to the source that the technique also discovers. Our traceroute server dataset includes reverse paths from servers in 186 networks back to 10 PlanetLab sources. [Key labels are in the same top-to-bottom order as the lines.]

and traceroute server measurements, the *Intersect TR* line in Figure 8 shows how well this approach works, by assuming the reverse path is the same as the forward path until it intersects one of the traceroutes [4]. No system today performs this type of path intersection on-demand for users. In the median (mean) case, this traceroute intersection shares 69% (67%) of the actual traceroute's hops. This result suggests that simply having a few hundred or thousand traceroute vantage points is not enough to reliably infer reverse paths; our system uses our novel measurement techniques to build off these traceroutes and achieve much better results.

What are the causes of differences between a reverse traceroute and a directly measured traceroute? Although it is common to think of the path given by traceroute as the true path, in reality it is also subject to measurement error. In this section, we discuss reasons traceroute and reverse traceroute may differ from each other and/or from the true path taken.

Assumptions of symmetry: When reverse traceroute is unable to identify the next reverse hop, we resort to assuming that hop is symmetric. These assumptions may lead to inaccuracies. For the PlanetLab dataset, if we consider only cases when we measure a complete path without assuming symmetry, in the median (mean) case reverse traceroute matches 93% (90%) of the traceroute alias-level hops. Similarly, for the traceroute server dataset, in the median (mean) case reverse traceroute finds 83% (81%) of the traceroute hops. We discuss how often we have to assume symmetry in Section 5.2.

Incomplete alias information: Many of the differences between the paths found by reverse traceroute and traceroute are due to missing alias information. Most alias-pair identification relies on sending probes to the two IP addresses and comparing the IP-IDs of the responses. For the PlanetLab dataset, of all the *missing* addresses seen in a traceroute that are not aliases of any hop in the corresponding reverse traceroute, 88% do not allow for such alias resolution [4]. Similarly, of all *extra* addresses seen in some reverse traceroute that are not aliases of any hop in the corresponding reverse traceroute, 82% do not allow for alias resolution. For the traceroute server dataset, 75% of the missing addresses and 74% of the extra ones do not allow it. Even for addresses that do respond to alias techniques, our alias sets are likely incomplete.

In these cases, it is possible or even likely that the two measurement techniques observe IP addresses that appear different but are in fact aliases of the same router. To partially examine how this lack of alias information limits our comparison, we use iPlane's Point-of-Presence (PoP) clustering, which maps IP addresses to PoPs defined by (AS,city) pairs [22]. For many applications such as diagnosis of inflated latencies [21], PoP level granularity suffices. iPlane has PoP mappings for 71% of the missing addresses in the PlanetLab dataset and 77% of the extra ones. For the traceroute server dataset, for which we have less alias information, it has mappings for 79% of the missing addresses and 86% of the extra ones. Figures 8 and 9 include *PoP-Level* lines showing the fraction of traceroute hops seen by reverse traceroute, if we consider PoP rather than router-alias-level comparison. In the median case, the reverse traceroute includes all the traceroute PoPs in both graphs (mean=94%, 84%). If reverse traceroute were measuring a different path than traceroute, then one would expect PoP-level comparisons to differ about as much as alias-level ones. The implication of the measured PoP-level similarity is that, when traceroute and reverse traceroute differ, they usually differ only in which router or interface in a PoP the path traverses. As a point of comparison, Figure 9 includes a PoP-level version of the *Guess Fwd* line; in the median (mean) case, it includes only 60% (61%) of the PoPs; the paths are quite asymmetric even at the PoP granularity.

[4] We omit the line from Figure 9 to avoid clutter.

Load-balancing and contemporaneous path changes: Another measurement artifact is that traceroute and reverse traceroute may uncover different, but equally valid, paths, either due to following different load-balanced options or due to route changes during measurement. To partly capture these effects, the *Next Day* lines in Figure 8 and 9 compare how many of the traceroutes' hops are also on traceroutes issued the following day. For the PlanetLab dataset, 26% of the paths exhibit some router-level variation from day to day. For the traceroute dataset, 49% of paths changed at the router level and (not shown in the graph) 15% changed at the PoP-level. In a loose sense, these results suggest an upper bound – even the same measurement issued at a different time may yield a different path.

Hidden or anonymous routers: Previous work comparing traceroute to record route paths found that 16% of IP addresses appear with RR but do not appear in traceroutes [36, 35]. *Hidden* routers, such as those inside some MPLS tunnels, do not decrement TTL. *Anonymous* routers decrement TTL but do not send ICMP replies, appearing as '*' in traceroute.

Exceptional handling of options packets: Packets with IP options are generally diverted to a router's route processor and may be processed differently than on the line card. For example, previous work suggests that packets with options are load-balanced per-packet, rather than per-flow [35].

An additional source of discrepancies between the two techniques is that traceroute and reverse traceroute make different assumptions about routing. Our techniques assume destination-based routing – if the path from D to S passes through R, from that point on it is the same as R's path to S. An options packet reports only links it actually traversed. With traceroute, on the other hand, a different packet uncovers each hop, and it assumes that if $R1$ is at hop k and $R2$ is at hop $k+1$, then there is a link $R1$–$R2$. However, it does not make the same assumption about destination routing, as each probe uses (S,D) as the source and destination. These differing assumptions lead to two more causes of discrepancies between a traceroute and a reverse traceroute:

Traceroute inferring false links: Although we use the Paris traceroute technique for accurate traversal of flow-based load balancers, it can still infer false links in the case of packet-based balancing [2]. These spurious links appear as discrepancies between traceroute and reverse traceroute, but in reality show a limitation of traceroute.

Exceptions to destination-based routing: With many tunnels, an option-enabled probe will see the entire tunnel as a single hop. With certain tunnels, however, our assumption of destination-based routing may not hold. When probed directly, an intermediate router inside the tunnel may use a path to the destination other than the one that continues through the tunnel. To partly capture the degree of this effect, we perform a study that eliminates it. From each of the 200 PlanetLab nodes used as destinations in this section, we issue both a traceroute and an RR ping to each of the 11 used as sources, so the RR ping will have the same source and destination as the traceroute (unlike with reverse traceroute's RR probes to intermediate routers). Since the RR slots may fill up before the probe reaches the destination, we only check if the traceroute matches the portion of the path that appears in the RR. After alias resolution, the median fraction of RR hops seen in the corresponding traceroute is 0.67, with the other factors described in this section accounting for the differences. This fraction is 0.2 lower than that for reverse traceroute, showing the difficulty in matching RR hops to traceroute hops.

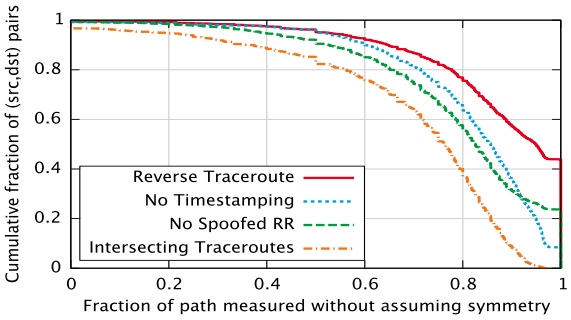


Figure 10: For the PlanetLab dataset, the fraction of reverse path hops measured, rather than assumed symmetric. The graph includes results with subsets of the reverse traceroute techniques.

5.2 Coverage

In Section 5.1, we noted that our paths are more accurate when our techniques succeed in measuring the entire path without having to fall back to assuming a link is symmetric. As seen in Figure 8, if forced to assume the entire path is symmetric, in the median case we would discover only 39% of the hops on a traceroute. In this section, we investigate how often our techniques are able to infer reverse hops, keeping us from reverting to assumptions of symmetry. Using the PlanetLab dataset, Figure 10 presents the results for our complete technique, as well as for various combinations of the components of our technique. The metric captured in the graph is the fraction of hops in the reverse traceroute that were measured, rather than assumed symmetric.

Reverse traceroute finds most hops without assuming symmetry. In the median path in the PlanetLab dataset, we measure 95% of hops (mean=87%), and in 80% of cases we are able to measure at least 78% of the path without assumptions of symmetric. By contrast, the traceroute intersection estimation technique from Figure 8 assumes in the median that the last 25% of the

path is symmetric (mean=32%). Although not shown in the graph, the results are similar for the traceroute server dataset – in the median case, reverse traceroute measures 95% of hops (mean=92%) without assuming symmetry.

The graph also depicts the performance of our technique if we do not use spoofed record route pings or do not use timestamping. In both cases, the performance drops off somewhat without both probing methods.

5.3 Overhead

We assess the overhead of our technique using the traceroute server dataset from Section 5.1, comparing the time and number of probes required by our system to those required by traceroute. The median (mean) time for one of the 10 PlanetLab sites to issue a traceroute to one of the 186 traceroute servers was 5 seconds (9.4 seconds). Using our current system, as available on `http://revtr.cs.washington.edu` and described in Section 4, the median (mean) time to measure a reverse traceroute was 41 seconds (116.0 seconds), including the time to send an initial forward traceroute (to determine if the destination is reachable and to present a round-trip path at the end). We have not yet pursued improving this aspect of the system. In the future, we will investigate lowering this delay by setting more aggressive timeouts for flaky PlanetLab vantage points, by cutting down on the communication overhead, and by attempting to adapt our probing rate to the rate limit of the particular target.

For each reverse traceroute measurement, our system sends the initial forward traceroute and a number of options-enabled ping packets, some of which may be spoofed. In cases when it is unable to determine the next reverse hop, it sends a forward traceroute and assumes the last hop is symmetric. In addition, we require traceroutes to build an atlas of paths to the source, and we use ongoing background mapping to identify adjacencies and to determine which vantage points are within RR-range of which prefixes.

If we ignore the probing overhead of the traceroute atlas and the mapping, in the median case, the only traceroute required is the initial one (mean=1.2 traceroutes). In the median (mean) case, a reverse traceroute requires 2 (2.6) record route packets, plus an additional 9 (21.2) spoofed RR packets. The median (mean) number of non-spoofed timestamp packets is 0 (5.1), and the median (mean) number of spoofed timestamp packets is also 0 (6.5). The median (mean) total number of options packets sent is 13 (35.4). As a point of comparison, traceroute uses around 45 probe packets on average, 3 for each of around 15 hops. At the end of a reverse traceroute, we also send 3 pings to each hop to measure latency. So, ignoring the creation of the various atlases, reverse traceroute generally requires roughly 2-3x more packets than traceroute.

IP address	AS name	Location	RTT
132.170.3.1	UCF	Orlando, FL	0ms
198.32.155.89	FloridaNet	Orlando, FL	0ms
198.32.132.64	FloridaNet	Jacksonville, FL	3ms
198.32.132.19	Cox Comm.	Atlanta, GA	9ms
68.1.0.221	Cox Comm.	Ashburn, VA	116ms
216.52.127.8	Internap	Washington, DC	35ms
66.79.151.129	Internap	Washington, DC	26ms
66.79.146.202	Internap	Washington, DC	24ms
66.79.146.241	Internap	Miami, FL	53ms
66.79.146.129	Internap	Seattle, WA	149ms

Table 1: Traceroute giving forward path from University of Central Florida to *66.79.146.129*.

The atlases represent the majority of our probe overhead. However, in many circumstances these atlases can be reused and/or optimized for performance. For example, if the source requests reverse paths for multiple destinations within a short period of time [45], we can reuse the atlas. As an optimization, we may need to only issue those traceroutes that are likely to intersect [22], and we can use known techniques to reduce the number of probes to generate the atlas [11]. We borrow the adjacency information needed for our timestamp probes from an existing mapping service [22]. To determine which spoofing vantage points are likely within record route range of a destination, we regularly issue probes from every spoofer to a set of addresses in each prefix. In the future, we plan to investigate if we can reduce this overhead by probing only a single address within each prefix.

6 Applications of Reverse Traceroute

We believe many opportunities exist for improving systems and studies using reverse traceroute. We next discuss three such examples of how reverse traceroute can be used in practice. We intend these sections to illustrate a few ways in which one can apply our tool; they are not complete studies of the problems.

6.1 Case study of debugging path inflation

Large content providers attempt to optimize client performance by replicating their content across a geographically distributed set of servers. A client is then redirected to the server to which it has minimum latency. Though this improves the performance perceived by clients, it can still leave room for improvement. Internet routes are often inflated [37], which can lead to round-trip times from a client to its nearest server being much higher than what they should be given the server's proximity. Using Google as an example, 20% of client prefixes experience more than 50ms latency over the minimum latency to the prefix's geographical region. Google wants a way to identify which AS is the cause of inflation, but it is hindered by the lack of information about reverse paths back to their servers from clients [21].

IP address	AS name	Location	RTT
66.79.146.129	Internap	Seattle, WA	148ms
66.79.146.225	Internap	Seattle, WA	141ms
137.164.130.66	TransitRail	Los Angeles, CA	118ms
137.164.129.15	TransitRail	Los Angeles, CA	118ms
137.164.129.34	TransitRail	Palo Alto, CA	109ms
137.164.129.2	TransitRail	Seattle, WA	92ms
137.164.129.11	TransitRail	Chicago, IL	41ms
137.164.131.165	TransitRail	Ashburn, VA	23ms
132.170.3.1	UCF	Orlando, FL	0ms
132.170.3.33	UCF	Orlando, FL	0ms

Table 2: Reverse traceroute giving reverse path from *66.79.146.129* back to University of Central Florida. The circuitous reverse path explains the huge RTT jump between the last two hops on the forward path. The third hop, 137.164.130.66 (internap-peer.lsanca01.transitrail.net), is a peering point between Internap and TransitRail in L.A.

As an illustration, we used reverse traceroute to diagnose an example of path inflation. We measured the RTT on the path from the PlanetLab node at the University of Central Florida to the IP address *66.79.146.129*, which is in Seattle, to be 149ms. Table 1 shows the forward path returned by traceroute, annotated with the locations of intermediate hops inferred from their DNS names. The path has some circuitousness going from Orlando to Washington via Ashburn and then returning to Miami. But, that does not explain the steep rise in RTT from 53ms to 149ms on the last segment of the path, because a hop from Miami to Seattle is expected to only add 70ms to the RTT[5].

To investigate the presence of reverse path inflation back from the destination, we determined the reverse path using reverse traceroute. Table 2 illustrates the reverse path, which is noticeably circuitous. Starting from Seattle, the path goes through Los Angeles and Palo Alto, and then returns to Seattle before reaching the destination via Chicago and Ashburn. We verified with a traceroute from a PlanetLab machine at the University of Washington that TransitRail and Internap connect in Seattle, suggesting that the inflation is due to a routing misconfiguration. Private communication with an operator at one of the networks confirmed that the detour through Los Angeles was unintentional. Without the insight into the reverse path provided by reverse traceroute, such investigations would not be possible by the organizations most affected by inflated routes.

6.2 Topology discovery

Studies of Internet topology rely on the set of available vantage points and data collection points. With a limited number available, routing policies bias what researchers measure. As an example, with traceroute alone, topology discovery is limited to measuring forward paths from a few hundred vantage points to each other and to other destinations. Reverse traceroute allows us to expose many peer-to-peer links invisible to traceroute.

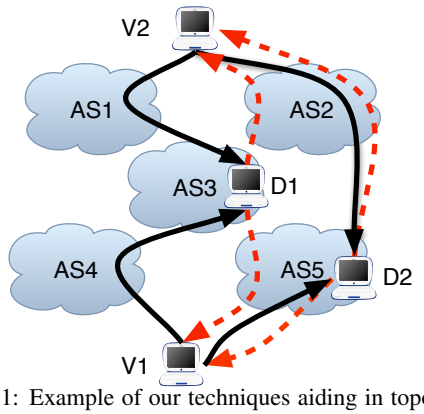


Figure 11: Example of our techniques aiding in topology discovery. With traceroutes alone, V1 and V2 can measure only the forward (solid) paths. If V2 is within 8 hops of D1, a record route ping allows it to measure the link AS3-AS2, and a record route ping spoofed as V1 allows it to measure AS3-AS5.

Figure 11 illustrates one way in which our techniques can uncover links. Assume that *AS3* has a peer-to-peer business relationship with the other ASes. Because an AS does not want to provide free transit, most routes will traverse at most one peer-to-peer link. In this example, traffic will traverse one of *AS3*'s peer links only if it is sourced or destined from/to *AS3*. *V1*'s path to *AS3* goes through *AS4*, and *V2*'s path *AS3* goes through *AS1*. Topology-gathering systems that rely on traceroute alone [22, 1, 34] will observe the links *AS1-AS3*, *AS4-AS3*, and *AS2-AS5*. But, they will never traverse *AS3-AS5*, or *AS3-AS2*, no matter what destinations they probe (even ones not depicted). *V2* can never traverse *AS1-AS3-AS5* in a forward path (assuming standard export policies), because that would traverse two peer-to-peer links. However, if *V2* is within 8 hops of *D1*, then it can issue a record-route ping that will reveal *AS3-AS2*, and a spoofed record route (spoofed as *V1*) to reveal *AS3-AS5*[6].

Furthermore, even services like RouteViews [27] and RIS [33], with BGP feeds from many ASes, likely miss these links. Typical export policies mean that only routers in an AS or its customers see the AS's peer-to-peer links. Since RouteViews has vantage points in only a small percentage of the ASes lower in the AS hierarchy, it does not see most peer links [29, 16].

To demonstrate how reverse traceroute can aid in topology mapping, we apply it to a recent study on mapping Internet exchange points (IXPs) [3]. That study used existing measurements, novel techniques, and thousands of traceroute servers to provide IXP peering matrices that were as complete as possible. As part of the

[5]Interestingly, the latency to Ashburn seems to also be inflated on the reverse path.

[6]Note that, because we only query for hops already known to be adjacent, our timestamp pings are not useful for topology discovery.

study, the researchers published the list of ASes they found to be peering at IXPs, the IXPs at which they peered, and the IP addresses they used in those peerings.

We measured the reverse paths back from those IP addresses to all PlanetLab sites. We discovered 9096 IXP peerings (triples of the two ASes and the IXP at which they peer) that are not in the published dataset, adding an additional 16% to the 58,534 peerings in their study. As one example, we increased the number of peerings found at the large London LINX exchange by 19%. If we consider just the ASes observed peering and not which IXP they were seen at, we found an additional 5057 AS links not in the 51,832 known IXP AS links, an increase of 10%. Of these AS links, 1910 do not appear in either traceroute [22] or BGP [40] topologies – besides not being known as IXP links, we are discovering links not seen in some of the most complete topologies available. Further, of the links in both our data and UCLA's BGP topology, UCLA classifies 1596 as Customer-to-Provider links, whereas the fact that we observed them at IXPs strongly suggests they are Peer-to-Peer links. Although the recent IXP study was by far the most exhaustive yet, reverse traceroute provides a way to observe even more of the topology.

6.3 Measuring one-way link latency

In addition to measuring a path, traceroute measures a round-trip latency for each hop. Techniques for geolocation [42, 18], latency estimation [22], and ISP comparisons [25], among others, depend on link latency measurements obtained by subtracting the RTT to either endpoint, then halving the difference (possibly with a filter for obviously wrong values). This technique should yield fairly accurate values if routes traverse the link symmetrically. However, previous work found that 88-98% of paths are asymmetric [15] resulting in substantial errors in link latency estimates [39]. More generally, the inability to isolate individual links is a problem when using network tomography to infer missing data – tomography works best only when the links are traversed symmetrically or when one knows both the forward and reverse paths traversed by the packets [10, 6].

A few alternatives exist for estimating link latencies but none are satisfactory. Rocketfuel infers link weights used in routing decisions [23], which may or may not reflect latencies. The geographic locations of routers provide an estimate of link latency, but such information may be missing, wrong, or outdated, and latency does not always correspond closely to geographic distance [18].

In this section, we revisit the problem of estimating link latencies since we now have a tool that provides reverse path information to complement traceroute's forward path information. Given path asymmetry, the reverse paths from intermediate routers likely differ from the end-to-end traceroutes in both directions. Without reverse path information from the intermediate hops back to the hosts, we cannot know which links a round-trip latency includes. Measurements to endpoints and intermediate hops yield a large set of paths, which we simplify using IP address clustering [22]. We then generate a set of linear constraints: for any intermediate hop R observed from a source S, the sum of the link latencies on the path from S to R plus the sum of the link latencies on the path back from R must equal the round-trip latency measured between S and R. We then solve this set of constraints using least-squares minimization, and we also identify the bound and free variables in the solution. Bound variables are those sufficiently constrained for us to solve for the link latencies, and free variables are those that remain under constrained.

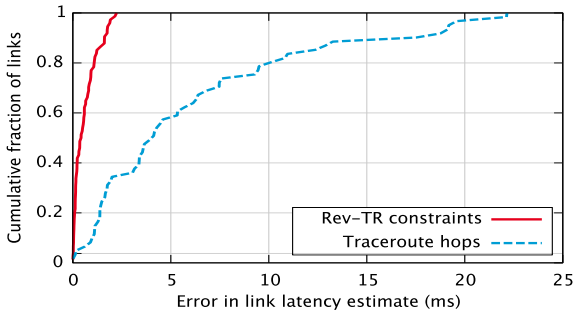


Figure 12: Error in estimating latencies for Sprint inter-PoP links. For each technique, we only include links for which it provided an estimate: 61 of 89 links using traceroute, and 74 of 89 using reverse traceroute. Ground truth reported only to 0.5ms granularity.

We evaluate our approach on the Sprint backbone network by comparing against inter-PoP latencies Sprint measures and publishes [38]. We consider only the directly connected PoPs and halve the published round-trip times to yield link latencies we use as ground truth, for 89 links between 42 PoPs. We observe 61 of the 89 links along forward traceroutes and 79 with reverse traceroute. We use these measurements to formulate constraints on the inter-PoP links, based on round-trip latencies measured from PlanetLab nodes to the PoPs using ping. This set of constraints allows us to solve for the latencies of 74 links, leaving 5 free and 10 unobserved.

As a comparison point, we use a traditional method for estimating link latency from traceroutes [22]. For each forward traceroute that traverses a particular Sprint link, we sample the link latency as half the difference between the round-trip delay to either end, then estimate the link latency to be the median of these samples across all traceroutes. Figure 12 shows the error in the latency estimates of the two techniques, compared to the published ground truth. Our approach infers link latencies with errors from 0ms to 2.2ms for the links, with a me-

dian of 0.4ms and a mean of 0.6ms. Because Sprint reports round-trip delays with millisecond granularity, the values we use for ground truth have 0.5ms granularity, so our median "error" is within the granularity of the data. The estimation errors using the traditional traceroute method range from 0ms to 22.2ms, with a median of 4.1ms and a mean of 6.2ms – $10x$ our worst-case, median, and mean errors. Based on this initial study of a single large network for which we have ground-truth, using reverse traceroute to generate and solve constraints yields values very close to the actual latencies, whereas the traditional approach does not.

7 Related work

Measurement techniques: Previous work concluded that too many paths dropped packets with IP options for options to form the basis of a system [13]. The Passenger and DisCarte projects, however, showed that the record route option, when set on traceroute packets, reduces false links, uncovers more routers, and provides more complete alias information [36, 35]. Hubble demonstrated the use of spoofed packets to probe a path in one direction without having to probe the other [19], but it does not determine the routers along the reverse path. Addressing this limitation in Hubble was part of the original motivation for this work.

The contributions of these various projects is in how they employ existing IP techniques – options and spoofing – towards useful ends. Our work employs the same IP techniques in new ways. We demonstrate how spoofing with options can expose reverse paths. Whereas Passenger and DisCarte used RR to improve forward path information, we use RR in non-TTL-limited packets to measure reverse paths. As far as we are aware, our work is the first to productively employ the timestamp option.

Techniques for inferring reverse path information: Various earlier techniques proposed methods for inferring limited reverse path information. Before such packets were routinely filtered, one study employed loose source-routing [31] to measure paths from numerous remote sites. Other interesting work used return TTL values to estimate reverse routing maps towards sources; however, the resulting maps contained less than half the actual links, as well as containing multiple paths from many locations [7]. PlanetSeer [43] and Hubble [19] included techniques for isolating failures to either the forward or reverse path; neither system, however, can give information about where on a reverse path the failure occurs. Netdiff inferred path asymmetry in cases where hop counts differ greatly in the two directions [25]; however, as our example in Section 6.1 shows, very asymmetric paths can have the same hop count. Tulip used ICMP timestamps (not the IP timestamp option we use) and other techniques to identify reordering and loss along either the forward or reverse path [24].

Systems that would benefit from reverse path information: Many systems seem well-designed to make use of reverse path information, but, lacking it, make various substitutions or compromises. We mention some recent ones here. Geolocation systems use delay and path information to constrain the position of targets [14, 18, 42], but, lacking reverse path data, are under constrained. iPlane shows that knowledge of a few traceroutes from a prefix greatly improves path predictions [22], but lacks vantage points in most. iSpy attempted to detect prefix hijacks using forward-path traceroutes, yet the signature it looked for is based on the likely pattern of reverse paths [46]. Similarly, intriguing recent work on inferring topology through passive observation of traffic bases its technique on an implicit assumption that the hop counts of forward and reverse paths are likely to be the same [12]. Similarly, systems for network monitoring often assume path symmetry [8, 25]. All these efforts can potentially benefit from the work described here.

8 Conclusion

Although widely-used and popular, traceroute is fundamentally limited in that it cannot measure reverse paths. This limitation leaves network operators and researchers unable to answer important questions about Internet topology and performance. To solve this problem, we developed a reverse traceroute system to measure reverse paths from arbitrary destinations back to the user. The system uses a variety of methods to incrementally build a path back from the destination hop-by-hop, until it reaches a known baseline path. We believe that our system makes a strong argument for both the IP timestamp option and source spoofing as important measurement tools, and we hope that PlanetLab and ISPs will consider them valuable components of future measurement testbeds.

Our reverse traceroute system is both effective – in the median case finding all of the PoPs seen by a direct traceroute along the same path – and useful. The tool allows operators to conduct investigations impossible with existing tools, such as tracking down path inflation along a reverse route. Many operators seem to view reverse traceroute as a useful tool – based on the results presented in this paper, we received requests to help us test the tool and offers of spoofing vantage points, including hosts at all the PoPs of an international backbone network. The system's probing methods have also proved useful for topology mapping. In illustrative examples, we demonstrated how our system can discover more than a thousand peer-to-peer links invisible to both BGP route collectors and to traceroute-based mapping

efforts, as well as how it can be used to accurately measure the latency of backbone links. We believe the accuracy and coverage of the tool will only improve as we add additional vantage points. A demo of our tool is available at http://revtr.cs.washington.edu.

Acknowledgments

We gratefully acknowledge John Byers, our shepherd, and the anonymous NSDI reviewers for their valuable feedback on earlier versions of this paper. Adam Bender, Rob Sherwood, Ricardo Oliveira, Brice Augustin, and Balachander Krishnamurthy provided access to and help with their tools and data. This work was funded partially by Google, Cisco, and NSF grants CNS-0905568 and MRI-0619836. We are thankful for their support.

References

[1] Archipelago measurement infrastructure. http://www.caida.org/projects/ark/.

[2] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. Avoiding traceroute anomalies with Paris traceroute. In *IMC*, 2006.

[3] B. Augustin, B. Krishnamurthy, and W. Willinger. IXPs: Mapped? In *IMC*, 2009.

[4] A. Bender, R. Sherwood, and N. Spring. Fixing Ally's growing pains with velocity modeling. In *IMC*, 2008.

[5] R. Beverly, A. Berger, Y. Hyun, , and k claffy. Understanding the efficacy of deployed Internet source address validation filtering. In *IMC*, 2009.

[6] T. Bu, N. Duffield, F. Presti, and D. Towsley. Network tomography on general topologies. In *SIGMETRICS*, 2002.

[7] H. Burch. *Measuring an IP network in situ*. PhD thesis, CMU, 2005.

[8] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. In *SIGCOMM*, 2004.

[9] D. Choffnes and F. E. Bustamante. Taming the torrent: A practical approach to reducing cross-ISP traffic in peer-to-peer systems. In *SIGCOMM*, 2008.

[10] M. Coates, A. Hero, R. Nowak, and B. Yu. Internet tomography. *IEEE Signal Processing Magazine*, 2002.

[11] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *SIGMETRICS*, 2005.

[12] B. Eriksson, P. Barford, and R. Nowak. Network discovery from passive measurements. In *SIGCOMM*, 2008.

[13] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica. IP options are not an option. Technical report, EECS Department, University of California, Berkeley, 2005.

[14] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of Internet hosts. *IEEE/ACM TON*, 2006.

[15] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker. On routing asymmetry in the Internet. In *Autonomic Networks Symposium in Globecom*, 2005.

[16] Y. He, G. Siganos, M. Faloutsos, and S. Krishnamurthy. A systematic framework for unearthing the missing links. In *NSDI*, 2007.

[17] Internet address hitlist dataset, PREDICT ID USC LANDER internet_address_hitlist_it28wbeta20090914. http://www.isi.edu/ant/lander.

[18] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. Towards IP geolocation using delay and topology measurements. In *IMC*, 2006.

[19] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson. Studying black holes in the Internet with Hubble. In *NSDI*, 2008.

[20] K. Keys, Y. Hyun, and M. Luckie. Internet-scale alias resolution with MIDAR. Under preparation, 2010.

[21] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving beyond end-to-end path information to optimize CDN performance. In *IMC*, 2009.

[22] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *OSDI*, 2006.

[23] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *IMW*, 2002.

[24] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet path diagnosis. In *SOSP*, 2003.

[25] R. Mahajan, M. Zhang, L. Poole, and V. Pai. Uncovering performance differences among backbone ISPs with Netdiff. In *NSDI*, 2008.

[26] Measurement Lab website. http://www.measurementlab.net/.

[27] D. Meyer. RouteViews. http://www.routeviews.org.

[28] NANOG 45, 2009.

[29] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang. In search of the elusive ground truth: The Internet's AS-level connectivity structure. In *SIGMETRICS*, 2008.

[30] Outages mailing list. http://isotf.org/mailman/listinfo/outages.

[31] J.-J. Pansiot and D. Grad. On routes and multicast trees in the Internet. *SIGCOMM CCR*, 1998.

[32] RIPE 58, 2009.

[33] RIPE RIS. http://www.ripe.net/ris/.

[34] Y. Shavitt and E. Shir. DIMES: Let the Internet measure itself. *SIGCOMM CCR*, 2005.

[35] R. Sherwood, A. Bender, and N. Spring. Discarte: A disjunctive internet cartographer. In *SIGCOMM*, 2008.

[36] R. Sherwood and N. Spring. Touring the Internet in a TCP sidecar. In *IMC*, 2006.

[37] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. In *SIGCOMM*, 2003.

[38] Sprint IP network performance. https://www.sprint.net/performance/.

[39] R. A. Steenbergen. A practical guide to (correctly) troubleshooting with traceroute. In *NANOG 45*, 2009. http://www.nanog.org/meetings/nanog45/presentations/Sunday/RAS_traceroute_N45.pdf.

[40] UCLA Internet topology collection. http://irl.cs.ucla.edu/topology/.

[41] http://www.merit.edu/mail.archives/nanog/. North American Network Operators Group mailing list.

[42] B. Wong, I. Stoyanov, and E. G. Sirer. Octant: A comprehensive framework for the geolocalization of Internet hosts. In *NSDI*, 2007.

[43] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. PlanetSeer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, 2004.

[44] Y. Zhang, Z. M. Mao, and M. Zhang. Effective diagnosis of routing disruptions from end systems. In *NSDI*, 2008.

[45] Y. Zhang, V. Paxson, and S. Shenker. The stationarity of Internet path properties: Routing, loss, and throughput. *ACIRI Technical Report*, 2000.

[46] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush. iSpy: detecting IP prefix hijacking on my own. In *SIGCOMM*, 2008.

Not only E.T. Phones Home: Analysing the Native User Tracking of Mobile Browsers

John Pegioudis
FORTH & University of Crete
Greece

Emmanouil Papadogiannakis
FORTH & University of Crete
Greece

Nicolas Kourtellis
Telefonica Research
Spain

Evangelos P. Markatos
FORTH & University of Crete
Greece

Panagiotis Papadopoulos
FORTH
Greece

ABSTRACT

Contemporary browsers constitute a critical component of our everyday interactions with the Web. Similar to a small, but powerful operating system, a browser is responsible to fetch and run web apps locally, on the user's (mobile) device. Even though in the last few years, there has been an increased interest for tools and mechanisms to block potentially malicious behaviours of web domains against the users' privacy (e.g., ad blockers, incognito browsing mode, etc.), it is still unclear if the user can browse the Web in private.

In this paper, we analyse the natively generated network traffic of 15 mobile browser apps under different configurations to investigate if the users are capable of browsing the Web privately, without sharing their browsing history with remote servers. We develop a novel framework (Panoptes) to instrument and monitor separately the mobile browser traffic generated by (a) the web engine and (b) natively by the mobile app. By crawling a set of websites via Panoptes, and analyzing the native traffic of browsers, we find that there are browsers (i) who persistently track their users, and (ii) browsers that report to remote servers (geolocated outside EU), the exact page and content the user is browsing at that moment. Finally, we see browsers communicating with third-party ad servers while leaking personal and device identifiers.

CCS CONCEPTS

• **Security and privacy** → **Web application security**; • **Information systems** → *Traffic analysis*; **Online advertising**.

KEYWORDS

Mobile Browsers, Native Traffic, User Tracking, Browser History

ACM Reference Format:
John Pegioudis, Emmanouil Papadogiannakis, Nicolas Kourtellis, Evangelos P. Markatos, and Panagiotis Papadopoulos. 2023. Not only E.T. Phones Home: Analysing the Native User Tracking of Mobile Browsers. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3618257.3624842

IMC '23, October 24–26, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0382-9/23/10.
https://doi.org/10.1145/3618257.3624842

1 INTRODUCTION

The proliferation of web applications has made contemporary browsers a critical component of our interactions with the Internet. Users use their browser as a gateway to the Web, accessing remote content and services. This functionality makes browsers' role similar to a small but powerful operating system where providers can instantly run their (web) applications locally, on the user's device.

Unfortunately, there are many examples of web providers either delivering malware or spying on their users via tracking their behavior [1–5]. Thus, in recent years, we see a great variety of methods, features [6–8] and products (e.g., ad-blocking extensions or browsers) aiming at detecting and blocking such malicious user tracking behaviors of websites and applications. Thus, *have users won this privacy war? If they block the tracking requests generated by the web engine, can they finally browse in private?*

In this study, we aim at investigating this exact question, motivated by the fact that considering its key role (acting as the user's gateway), the browser can *know what content the user browses and when.* Consequently, there is a hyper-concentration of personal information in the browser (even greater in recent years, as more than 58% of the web traffic comes from the more personal, mobile devices [9]), that enables the browser to accurately and fully estimate the preferences and interests of the user, at any time.

Towards this goal, we develop a first of its kind framework to instrument mobile browsers and monitor the outgoing traffic of both the web engine and the mobile browser app itself. Hence, we are able to analyse the behavior of (i) the websites when rendered in the web engine but more importantly (ii) the browser's, when communicating with second or third-party web entities, or the remote server of the browser vendor (via the so called "phone home" requests).

In summary, the contributions of this study include:

(1) We propose Panoptes: a framework to instrument and monitor separately the mobile browser traffic which is generated by (i) the web engine and (ii) natively by the mobile app.
(2) We implement our framework and by instrumenting 15 different mobile browsers, we crawl 1000 websites. We provide our tool open source to aid the community with mobile browser auditing[1].
(3) By analyzing the captured traffic, we see that natively generated traffic can be as high as 1/3 of the total generated traffic. We also see that when 3 of the tested browsers "phone home", they report to servers located outside the EU (where

[1]Source code of Panoptes: https://github.com/BanForFun/panoptes

the crawling was performed), the exact page and content the user is browsing. One browser, in particular, does so together with a persistent identifier so users can be tracked even if they use Tor [10] or a proxy. In addition, we see 3 other browsers communicating with ad servers or Facebook Graph API while leaking user personal information and other device-specific identifiers.

2 BROWSER TRAFFIC MONITORING

To better understand the behavior of mobile browsers and their actions, we need to monitor their network traffic while we automate their crawling campaigns. The easiest way to achieve this would be via instrumenting browser apps installed in emulated devices. However, this is something that could alter the behavior of websites (i.e., non-realistic web visits) and harm the reproducibility of the crawls [11].

To that extent, we develop Panoptes: the first of its kind framework to instrument instances of various browser applications residing in a physical device and monitor their outgoing network traffic. Our framework is able to capture and differentiate the network traffic generated by the web engine, from the traffic that the browser app natively creates. Panoptes automatically initiates browser instances, visits websites and captures their network traffic.

As a testbed, we utilize a Linux Desktop responsible for the instrumentation of mobile browsing applications and an Android SM-T580 Samsung Galaxy Tablet, running Android 11[2], that hosts the mobile apps, and registers and stores network traffic. In Figure 1, we provide a high level overview of our methodology. To instrument browser instances and automate operations, we use the UI automator of Appium [13], Frida [14] instrumentator and Chrome DevTools Protocol (CDP) [15]. The implementation of our framework is publicly available.

2.1 Crawling campaigns

Before starting every crawling campaign, we reset the browser application to its default factory settings using Appium. Then, we start each browser using Frida and go through the setup wizard manually to test various configurations. To visit a website, we employ CDP and instrument the page object to navigate to a specific domain. For browsers that do not support CDP, we hook into the WebView's functions using a custom Frida script and instrument them accordingly.

To ensure that the auto-complete feature of modern browsers will not pollute our network traces, we navigate to every website directly by using CDP or Frida without typing the domain on the address bar. When we visit the website, we consider that it is ready when the `DOMContentLoaded` event has been triggered, or when 60 seconds have passed since the visit started. Then, we wait for an additional period of 5 seconds to ensure that the website has completely loaded and that all its operations have taken place.

[2]While our work focuses on Android, similar behavior is expected to appear by the same browsers while running on iOS, since past findings have indicated so [12]. Verifying that is part of our planned future work.

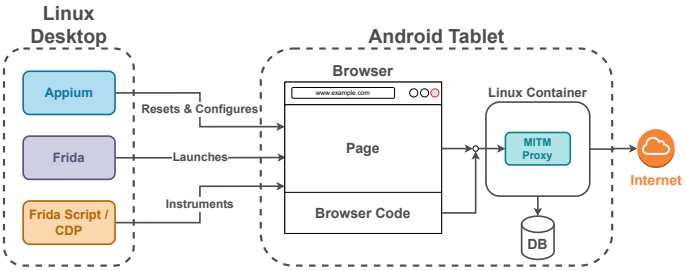


Figure 1: Overview of our framework system design.

Table 1: Our dataset of mobile browsers along with their version number.

Browser	Version	Browser	Version
Chrome	113.0.5672.77	**DuckDuckGo**	5.158.0
Edge	113.0.1774.38	**Dolphin**	12.2.9
Opera	75.1.3978.72329	**Whale**	2.10.2.2
Vivaldi	6.0.2980.33	**Mint**	3.9.3
Yandex	23.3.7.24	**Kiwi**	112.0.5615.137
Brave	1.51.114	**CocCoc**	117.0.177
Samsung	20.0.6.5	**UC International**	13.4.2.1307
QQ	13.7.6.6042		

2.2 Monitoring encrypted browser traffic

To monitor and capture encrypted network traffic, Panoptes utilizes a Man-In-The-Middle (MITM) proxy. Specifically, in the Android tablet, a Debian container with mitmproxy [16] is installed (along with its CA certificate) in transparent mode. We use this mode to proxy all traffic at the network layer without having to configure each browser app[3].

For every browser we analyze, Panoptes extracts their unique kernel UID under which each browser process is running [17] to create *iptable* rules and divert their traffic through the proxy. In addition to this, Panoptes creates rules to block all HTTP/3 traffic, as at the time of crawling, mitmproxy did not support the QUIC protocol. This should not affect the behavior of crawled websites, since they also support older HTTP versions which the browsers automatically fall back to.

2.3 Splitting native & web engine browser traffic

A very important contribution of Panoptes is its ability to differentiate between traffic that (i) was generated in the web engine by the website itself as part of its operations, or due to the actions of the user, and (ii) the traffic generated natively by the browser app itself, as designed by the browser vendor (i.e., native traffic). To achieve this, we make use of CDP and for each browser instance, we intercept all HTTP requests initiated by the website. In the case of the *UC International* browser, we use Frida to hook into an internal API. Then, for each intercepted request, we perform tainting by piggybacking an additional custom HTTP header using the 'x-' prefix that does not interfere with existing headers.

[3]Using a MITM proxy, can cause issues in apps that use certificate pinning by preventing them from issuing some requests (to pinned domains). In this paper, we make no attempt to bypass certificate pinning, and we consider our results as the lower bounds of the leaking that may happen in these browsers.

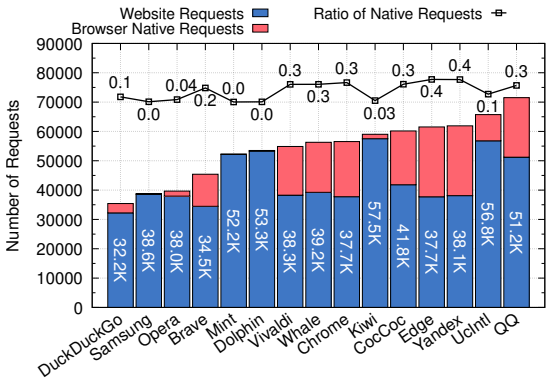


Figure 2: Number of requests generated by the website and the browser. In the cases of Edge and Yandex the ratio of natively generated requests of the brower app (red) to the requests generated by the web engine (blue) can reach as high as 0.38 and 0.39 respectively.

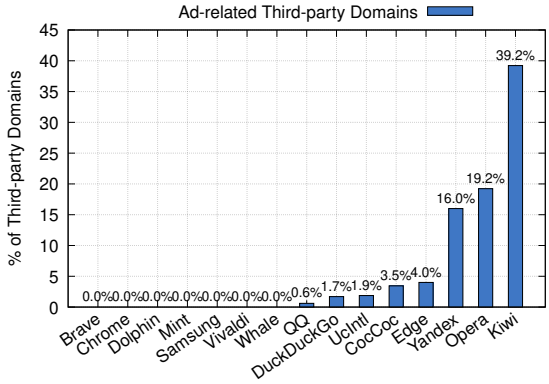


Figure 3: The percentage of domains that browsers send native request being third party and ad related.

Considering that all the network traffic of the Android tablet flows through the MITM proxy, in Panoptes, we have developed a custom MITM add-on to inspect all headers and separate the tainted ones. Thus, when HTTP requests arrive at the proxy, the MITM addon intercepts them at runtime, filters the tainted ones (i.e., requests originated from the website) before removing the additional (custom) header and forwarding them to their original destination. If a request is not tainted, it means that the request was generated natively by the browser app. The two different categories of the requests are finally stored in different local databases.

3 DO BROWSERS TRACK THEIR USERS?

By using the above methodology, we investigate our motivating question: *Can users browse without being tracked*?

Data collection: Hence, we download and install the top [18] 15 mobile browser apps[4] from Google Play Store (see Table 1 for details). Additionally, we (i) obtain the top 500 most popular websites based on the Tranco list [20] and (ii) collect an extra 500 websites that are associated with sensitive information based on the Curlie directory [21]. To achieve the later, from the Curlie directory, we manually select websites associated with sensitive issues regarding Society (e.g., warfare and conflict), Religion, Sexuality and Health (e.g., mental health). Hence, we form a set of 1000 websites [22], and we use Panoptes to visit each of these sites via each of the mobile browsers in our dataset. Our crawls were performed from an EU-based vantage point.

3.1 Natively generated network traffic

Next, we set out to explore the volume of traffic generated natively by the browser app. In Figure 2, we plot for each browser, the number of requests generated (i) by all visited websites (blue) and (ii) the number of requests the browser natively generated (red). Additionally, we compute in the plot (black line) the ratio of the total network traffic the natively generated requests of the browser app amount to. We see that a lot of browsers generate a substantial amount of network traffic that is not related to the website the user has visited. Specifically, for browsers like Vivaldi, Whale, CocCoc, Edge and Yandex, more than 1/3 of the total network traffic (as high as 0.39 and 0.38 in case of Edge and Yandex, respectively) is attributed to native requests that the browser sends to a remote server. It is important to note that this amount of additional (native) traffic is not related at all to the content the user is interested in and navigated to. As measured in the past [23, 24], such unsolicited network traffic consumes system resources and energy from the user's device.

Interestingly, as depicted in Figure 3, 8 of the browsers in our dataset issue native requests to third-party (ad) servers, as classified by the popular Steven Black host list [25]. Surprisingly, almost 40% of the distinct domains to which Kiwi sends native requests to are ad or analytics-related domains, including *rubiconproject.com*, *adnxs.com*, *openx.net*, *pubmatic.com*, *bidswitch.net*, *demdex.net*, etc.. Similarly, 19.2% and 16% of the domains with which Opera and Yandex communicate, respectively, are ad or analytics-related, including *appsflyersdk.com* and *doubleclick.net*. We also see CocCoc and Edge browsers sending native requests to analytics services like *adjust.com*, whereas it is interesting to note that CocCoc browser is an ad-blocking browser that enforces the easylist filterlist in its web engine [26].

In Figure 4, we plot the volume of outgoing network traffic (i.e., HTTP(S) requests) generated by both the browser and the websites. We discover that some browsers generate a substantial amount of traffic. Specifically, in the case of the QQ browser, we see that the additional traffic generated by the browser app can amount to a staggering 42% extra traffic. Of course, such unsolicited and unnecessary traffic can have considerable impact on the user's data plan and performance.

3.2 Leaking the user browsing history

As a next step, we analyze the information that the natively-issued (from within the browser app) HTTP(S) requests report to remote servers. When such remote servers are controlled by the browser vendor or a subsidiary of theirs, we call these generated requests *"phone-home"* requests.

[4]We exclude Firefox from our study because it supports different instrumentation protocols [19] which are incompatible with Panoptes.

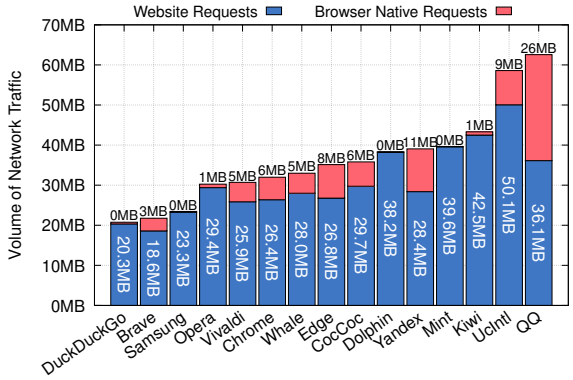


Figure 4: Volume of network traffic generated by (blue) the website and (pink) the browser itself natively. In the case of QQ browser, the additional traffic generated by the browser app can amount to an extra 42%.

```
POST https://s-odx.oleads.com/api/v1/sdk_fetch
body: {"channelId":"adxsdk_for_opera_ofa_final","countryCode":"ANONYMIZED",
"availableServices":["GOOGLE_PLAY"],"languageCode":"EN","appPackageName":"com
.opera.browser","appVersion":"75.1.3978.72329","sdkVersion":"1.12.2","deviceType":
"PHONE","osType":"ANDROID","osVersion":"11","deviceVendor":"Samsung","device
Model":"SM-T580","deviceScreenWidth":1200,"deviceScreenHeight":1920,"operaId":
"7e5d1382f2dd484e9d036519c8a908ddd5de945b100bc9e66582e2ed4ab0b2ab","operator":"",
"connectionType":"WIFI","userConsent":"false","duration":0,"latitude":ANONYMIZED,
"longitude":ANONYMIZED,"positionTimestamp":1683927615,"placementKey":"s569498648
9856","adCount":2,"floorPriceInCent":0,"timestamp":1683927615,"token":"e4818505
a103a7fc3b3e74174c37e570","supportedAdTypes":["SINGLE"],"supportedCreativeTypes":
["BIG_CARD","DISPLAY_HTML_300x250","LEADS","NATIVE_NEWSFLOW_1_IMAGE","NATIVE_
NEWSFLOW_3_IMAGES","POLL","PREBID_INTERSTITIAL","PREBID_NATIVE","SURVEY_SINGLE_
CHOICE","SURVEY_MULTIPLE_CHOICE","SURVEY_FEW_QUESTIONS","VAST_3_URL","VAST_3_XML",
"NATIVE_VAST","VIDEO_16x9"],"supportedFeaturesList":[{"cache":["v1"]},{"video":
["novast"]},{"display":["mrect","mraid2"]}],"omidpn":"Opera","omidpv":
"omsdk-1.3.16-Opera"}:
```

Listing 1: Native ad request issued by Opera

The Yandex case: By parsing the traffic generated by the Yandex browser app, to our surprise, we see that the browser explicitly leaks the user's browsing history via *"phone-home"* requests to its vendor's servers. Specifically, we find that for each page visited, the browser sends a native request towards *sba.yandex.net* with the visited URL (Base64) encoded in the request parameters, along with a second request towards *api.browser.yandex.ru* with the visited hostname, and a unique identifier of the user. This means Yandex company can track the user persistently even if they erase cookies, or change their IP address or use Tor/anonymous proxy or VPN!

To add insult to injury, these user tracking requests are not being issued only the first time the user visits a website, but on each and every page visit. This means that *the remote server knows exactly what and when the user browses it*, thus leaking temporal and behavioral patterns of the user: i.e., when they are online, how many times they have visited a web page, and how much time they spent there before they browse to a different website.

At this point, it is important to note that by reporting the full URL, the remote server has access not only to the visited domain, but also the full list of URL parameters, and thus, the exact content the user consumes (and in some cases, even what actions they performed on the visited website). As an example, for a user browsing a video on YouTube, the remote server learns not only that they visited YouTube, but also the specific video they browsed and when this visit happened. It is easy to anticipate that such a granular access to the user's history leaks interests, sexual preferences, political beliefs, etc.

What about other browsers? While for Yandex there were past reports [12] indicating potential mishandling of personal data (such as device, network and IP address), the users do not seem to be able to browse without being tracked on other browsers as well. Indeed, as a next step, we analyze the network traffic that all 15 browser apps in our dataset generate. Surprisingly, we see that apart from Yandex, the browsers of QQ and UC International also send the entire visited URL via "phone home" requests, including the full path and the query parameters, thus leaking the full content the user is browsing. Note that, contrary to the other two browsers, UC International does not leak the browser history by issuing native requests to the remote server. Instead, it injects an (obfuscated) JavaScript snippet into every web page the user browses. In the generated requests, UC International leaks to the remote server the user's city-level geolocation and ISP. In addition, we see Edge and Opera browsers reporting every single visited domain to Bing API and Opera Sitecheck (Opera's anti-phishing service), respectively. In addition, 8 out of all 15 mobile browsers in our dataset query Cloudflare's or Google's third-party DNS-over-HTTPS services for the visited domains with the rest (7) of them using the device's local DNS stub resolver.

Incognito mode: Next, we investigate if a privacy-conscious user has the option to stop browsers from leaking their browsing history by browsing in incognito mode. Specifically, we use Panoptes with the incognito modes of Edge, UC International and Opera to crawl a subset of the websites in our dataset[5]. We find that these browsers that leak the browsing history of their users, continue to do so, no matter what mode the user is browsing on. This highlights a gap between what the user expects to happen when they open a browser in incognito mode and what happens in reality.

Reporting visits to sensitive content: It has been shown in the past [27] that sharing sensitive information may lead to the exposure of the user's political beliefs, sexual preferences, etc., and thus, to targeted attacks (e.g., smear campaigns, de-anonymization on social media [28], blackmailing, spear phising [29], etc.). Therefore, as a next step, we set out to explore if the browsers that share the browsing history of the user with remote servers, perform any sort of filtering locally to avoid leaking potential user visits to sensitive content (as per the sensitive categories that Google Ads block [30]). Specifically, we use the Panoptes framework to crawl websites dealing with content from these sensitive categories (i.e., religion, sexual preferences, political beliefs and health). We find that these same browsers (i.e., Yandex, UC International and QQ) continue to leak the entire URL the users visits, and thus, the exact content they browse.

3.3 Leaking PII and device identifiers

Next, we use keyword matching (via regex) and heuristics to extract potential Personally Identifying Information (PII) and device-specific information the browsers may leak via the URL parameters of the natively generated requests. We exclude the Android version

[5]Unfortunately the rest of the browsers that leaked users' browsing history (Yandex and QQ) do not provide an incognito mode.

Table 2: Personally Identifying Information (PII) and device-specific information leaked by the various browser apps. Connection type can be Metered or Unmetered while the Network type can be WiFi or Cellular.

Browser	Device Type	Device Manuf.	Timezone	Resolution	Local IP	DPI	Rooted Status	Locale	Country	Location (lat & long)	Connection Type	Network Type
Chrome	No	No	No	No	No	No	No	No	No	No	No	No
Edge	No	Yes	Yes	Yes	No	No	No	Yes	No	No	Yes	Yes
Opera	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	Yes
Vivaldi	No	No	No	Yes	No	No	No	No	No	No	No	No
Yandex	Yes	Yes	No	Yes	No	Yes	No	Yes	No	No	No	Yes
Brave	No	No	No	No	No	No	No	No	No	No	No	No
Samsung	No	No	No	No	No	No	No	Yes	No	No	No	No
DuckDuckGo	No	No	No	No	No	No	No	No	No	No	No	No
Dolphin	No	No	No	No	No	No	No	No	No	No	No	No
Whale	No	No	No	Yes	Yes	No	Yes	Yes	Yes	No	No	Yes
Mint	No	No	Yes	Yes	No	No	No	Yes	Yes	No	No	No
Kiwi	No	No	No	No	No	No	No	No	No	No	No	No
CocCoc	Yes	Yes	No	Yes	No	No	No	Yes	Yes	No	No	No
QQ	Yes	Yes	No	Yes	No	No	No	No	No	No	No	No
UC Int.	No	No	No	No	No	No	No	Yes	No	No	No	Yes

and the device model from our analysis, as such information is reported by default for compatibility purposes by all vendors through the HTTP User-Agent header.

In Table 2, we summarize our findings. Interestingly, we see the Whale browser leaking the local IP along with the rooted status of the device, its network type and the country-level geolocation of the user. Similar geolocation information is shared also by Mint, CocCoc and Opera browsers, with the latter browser sharing also the longitude and latitude coordinates of the user. It is important to note that the information leaked by Opera and QQ is not shared with their vendors, but with ad servers, as described earlier in Section 3.1. An example of such a native request along with the information shared can be seen in Listing 1.

3.4 International data transfers

The General Data Protection Regulation (GDPR) imposes restrictions on the transfer of personal data outside the European Union (EU), to third-party countries or international organizations [31]. To understand where the remote servers (the browsers communicate with when they "phone home") are located, we follow past approaches [32]. In particular, we extract the IP address of every remote server receiving native requests from the tested browsers, and use a popular IP-to-geolocation service [33] to extract its country-level location. We see that while the crawls took place from EU, in case of the mobile browsers Yandex, QQ and UC International which leak in full detail the browsing history of the users, the requests are being received by servers located in Russia, China, and Canada, respectively. This finding exposes a misalignment with the users' expectation regarding online privacy [3, 34], (especially in incognito or "private" mode). Indeed, it is not clear that users are fully aware that their data may end up in countries outside the EU.

3.5 Phoning home when idle

Finally, we examine the native network activity of all browsers in our dataset during idle time. Using Panoptes, we launch each browser and leave them idle (at the start page) without any user interaction for 10 minutes, while monitoring their network traffic. As we can see in Figure 5, even though there is no web activity, the majority of the mobile browsers have frequent communication with their associated remote servers. This communication grows exponentially within the first minute (mostly to update favicons, thumbnails and DNS entries of the websites in the start page), before they reach a relative plateau where they "phone home" to query for updates and send telemetry and analytics). Opera follows a linear growth because of the frequently updated Opera News feed appearing in the starting page.

Yet, we see cases like Dolphin and Mint sending 46% and 8%, respectively, of their native requests to Facebook's Graph APIs. Similarly, CocCoc sends 6.7% of its native request to adjust.com advertisers, whereas Edge is quite active communicating with various different domains, including the second parties msn, microsoft.com, bing.com, but also the third party advertisers and analytics of adjust.com, outbrain.com, zemanta.com and scorecardresearch.com. Finally, we see Opera sending 21.9% and 1.7% of its native requests to doubleclick.net and appsflyers.com, respectively.

4 RELATED WORK

Similar to our study, in [35] authors investigate the "phone home" functionality of the 6 most popular browsers along with its privacy implications. Specifically, they capture the traffic of the browser's web engine and analyze the device and user IDs leaked. Contrary to our work, authors focus on identifiers leaked during specific operations (like browser startup, search auto-complete and update of browser extensions), and show how these identifiers can be used to identify browser instances. In this work, we follow an automated and scalable approach using the Panoptes framework, contrary to the manual analysis of network traffic performed in [35]. Also, we discover that the user's browsing history is explicitly leaked by the browsers, and not because of the auto-complete feature.

Concurrently with our work, in [36] the authors studied the privacy implications of over 400 Android browsing applications, and found that some of these apps not only contain advertising and tracking libraries, but they also leak personal information to third parties. In this study, we consciously select a smaller set of popular browsers that account for 75% of the mobile browser market share worldwide [37]. Our approach allows us to dive deeper in investigating aspects that play a major role in understanding the extent of the problem. First, we perform our study from an EU vantage point (where GDPR is in effect) and show that important and private user

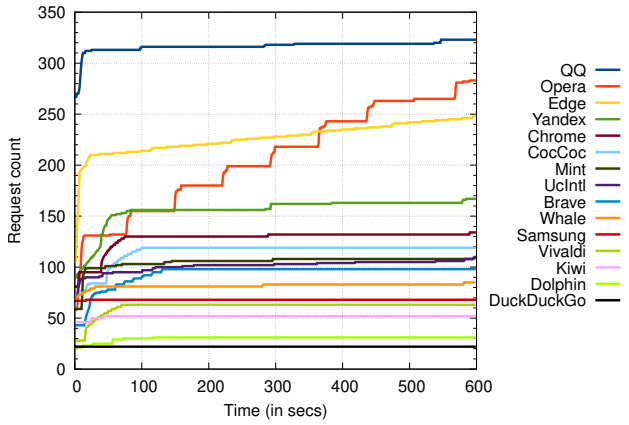


Figure 5: Timeline of native requests while the browser is idle for 10 minutes. The activity of most browsers grows exponentially within the first minute (mostly to update favicons, thumbnails and DNS entries of the websites in the start page), before they reach to a relative plateau. where they "phone home" to query for updates, send telemetry and analytics.

data, such as access to medical sites, is sent to servers outside the EU. Second, we investigate both regular browsing and incognito mode, and show that even in incognito mode some browsers leak the browsing history of their users, exposing the false sense of privacy that incognito mode may give to the users. This highlights the discrepancy between what the users expect to happen when they browse "anonymously" (i.e., incognito mode), and what mobile browsers actually do. Finally, we study separately domain name leaking and full path leaking. Indeed, the latter is more significant as it allows Web entities to have a clear understanding of their users' preferences and actions.

In [38] authors study the native requests of Safe Browsing functionality of Google Chrome and Yandex. They show that Safe Browsing can potentially be used as a tool to track specific classes of individuals. Additionally, they show that data included in Google and Yandex Safe Browsing provide a concrete set of URLs/domains that can be re-identified without much effort. In [39] authors analyzed private browsing modes in modern browsers and discussed their success at achieving the desired security goals. They point out several weaknesses in existing implementations, with the most severe ones enabling a local attacker to completely defeat the benefits of private mode. In [40] authors propose a new whitelist-based browsing mode that aims to act as a middle-ground between regular browsing and private mode, thus, enabling users to get the best of both worlds: the privacy of private mode, along with the convenience of the regular browsing mode.

Countermeasures: Considering that the user tracking performed by the browser app takes place natively, and gets leaked to remote servers via network requests of the browser app itself, traditional tracker/ad-blocking extensions cannot constitute a useful countermeasure. In [41] authors propose NoMoAds which leverages the mobile device's network interface as a universal vantage point to intercept, inspect, and block outgoing packets from mobile apps. NoMoAds extracts features from packet headers and/or payload to train machine learning classifiers for detecting native ad requests. Similarly, in [42] authors propose a cross-platform system (ReCon) that inspects apps' native traffic and leverages machine learning to reveal potential PII leaks, thus, giving mobile users control of what is shared with remote servers. In [43] authors study and compare the corresponding privacy implications when the user accesses an online service via the web, or via its dedicated mobile (native) app. Finally, they propose an anti-tracking mechanism to intercept and block third party native requests of mobile apps on the OS level based on custom filterlists.

5 SUMMARY AND CONCLUSION

In this work, we study the privacy implications of mobile browser apps and investigate whether users can truly browse the Web in private, even if they do block tracking requests of the web engine. To achieve this, we develop Panoptes, a framework to instrument instances of Android browser applications and monitor separately the mobile browser traffic which is generated by (i) the web engine and (ii) natively by the mobile browser app.

We use Panoptes to study 15 of the most popular browser applications and we collect data by crawling 1000 websites. As a summary of our findings, in this study we:

(1) Analyse the amount of native requests the mobile browsers generate, and find that it can amount to as high as 1/3 of the total generated traffic.
(2) See that when Yandex, QQ and UC International browsers "phone home", they report to their remote servers the exact page and content the user is browsing at real time.
(3) Find that Yandex, in particular, does this reporting together with a persistent identifier, so that users can be tracked even if they use Tor, an anonymous proxy, or a VPN.
(4) Confirm that the browser history leaking behavior happens (i) even when the users are browsing in incognito mode, or (ii) even when the users browse content of sensitive categories related to religion, sexual or political preferences, health issues etc.
(5) Find that, in some browser cases, these browsing history leaks are transmitted to servers located outside EU.
(6) See Opera, CocCoc, Dolphin and Mint browsers communicating with third-party ad and analytics servers, while they are also leaking PII and device-specific identifiers.

6 ACKNOWLEDGEMENTS

This project has received support from the European Union's Horizon 2020 Research and Innovation program under the CONCORDIA project (Grant Agreement No. 830927) and SPATIAL project (Grant Agreement No. 101021808). The authors bear the sole responsibility for the content presented in this paper, and any interpretations or conclusions drawn from it do not reflect the official position of the European Union nor the Research Innovation Foundation.

REFERENCES

[1] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, CCS'16, 2016.

[2] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos Markatos. Cookie synchronization: Everything you always wanted to know but were afraid to ask. In *The World Wide Web Conference*, WWW'19, 2019.

[3] Emmanouil Papadogiannakis, Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P. Markatos. User tracking in the post-cookie era: How websites bypass gdpr consent to track users. In *Proceedings of the Web Conference 2021*, WWW '21, page 2130–2141, 2021.

[4] Peter Snyder, Soroush Karami, Arthur Edelstein, Benjamin Livshits, and Hamed Haddadi. Pool-party: Exploiting browser resource pools as side-channels for web tracking. *arXiv preprint arXiv:2112.06324*, 2021.

[5] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P Markatos. Exclusive: How the (synced) cookie monster breached my encrypted vpn session. In *Proceedings of the 11th European Workshop on Systems Security*, pages 1–6, 2018.

[6] Google Developer. Safe browsing apis (v4). https://developers.google.com/safe-browsing/v4, 2023.

[7] Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. {WebGraph}: Capturing advertising and tracking information flows for robust blocking. In *31st USENIX Security Symposium*, USENIX Sec'22, 2022.

[8] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy*, SP'21, 2021.

[9] Josh Howarth. Internet traffic from mobile devices (apr 2023). https://explodingtopics.com/blog/mobile-internet-traffic, 2023.

[10] Tor Project. Tor project. https://www.torproject.org/, 2023.

[11] Jordan Jueckstock, Shaown Sarker, Peter Snyder, Aidan Beggs, Panagiotis Papadopoulos, Matteo Varvello, Benjamin Livshits, and Alexandros Kapravelos. Towards realistic and reproducibleweb crawl measurements. In *Proceedings of the Web Conference 2021*, WWW '21, page 80–91, 2021.

[12] Patrick McGee. Russian tech giant yandex's data harvesting raises security concerns. https://www.ft.com/content/c02083b5-8a0a-48e5-b850-831a3e6406bb, 2022.

[13] Appium. http://appium.io/docs/en/2.0/, 2023.

[14] Frida. https://frida.re/, 2023.

[15] Curlie. Chrome developers. https://chromedevtools.github.io/devtools-protocol/, 2023.

[16] Mitmproxy Project. mitmproxy - an interactive https proxy. https://mitmproxy.org/, 2023.

[17] Android Developers. Process. https://developer.android.com/reference/android/os/Process.html#myUid%28%29, 2023.

[18] statcounter. Top mobile browsers. https://gs.statcounter.com/browser-market-share/mobile, 2023.

[19] Firefox. Remote protocols. https://firefox-source-docs.mozilla.org/remote/index.html, 2023.

[20] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, February 2019.

[21] Curlie. The collector of urls. https://curlie.org/, 2023.

[22] John Pegioudis. Website list. https://github.com/BanForFun/panoptes-results/blob/master/1k.txt, 2023. Accessed on September 18th, 2023.

[23] Jiaping Gui, Stuart Mcilroy, Meiyappan Nagappan, and William G. J. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 100–110, 2015.

[24] Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P. Markatos. The cost of digital advertisement: Comparing user and advertiser views. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 1479–1489, 2018.

[25] Steven Black. Adware & malware hosts. https://github.com/StevenBlack/hosts, 2023.

[26] Coc Coc Browser. The best browser with adblocker. https://coccoc.com/en/chan-quang-cao, 2023.

[27] Artur Janc and Lukasz Olejnik. Web browser history detection as a real-world privacy threat. In *Computer Security – ESORICS 2010*, pages 215–231, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[28] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Christopher Kruegel. A practical attack to de-anonymize social network users. In *2010 IEEE Symposium on Security and Privacy*, pages 223–238, 2010.

[29] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, page 581–590, New York, NY, USA, 2006. Association for Computing Machinery.

[30] Google Ad Manager. Block sensitive categories. https://support.google.com/admanager/answer/2541069, 2023.

[31] Data Protection Commision. Transfers of personal data to third countries or international organisations. https://www.dataprotection.ie/en/organisations/international-transfers/transfers-personal-data-third-countries-or-international-organisations, 2018.

[32] Costas Iordanou, Georgios Smaragdakis, Ingmar Poese, and Nikolaos Laoutaris. Tracing cross border web tracking. In *Proceedings of the internet measurement conference 2018*, pages 329–342, 2018.

[33] Brand Media, Inc. Ip address lookup | geolocation. www.iplocation.net, 2023.

[34] Célestin Matte, Nataliia Bielova, and Cristiana Santos. Do cookie banners respect my choice?: Measuring legal compliance of banners from iab europe's transparency and consent framework. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 791–809. IEEE, 2020.

[35] Douglas J. Leith. Web browser privacy: What do browsers say when they phone home? *IEEE Access*, 9:41615–41627, 2021.

[36] Amogh Pradeep, Alvaro Feal, Julien Gamba, Ashwin Rao, Martina Lindorfer, Narseo Vallina-Rodriguez, and David Choffnes. Not your average app: A large-scale privacy analysis of android browsers. *Proceedings on Privacy Enhancing Technologies*, 1:29–46, 2023.

[37] StatsCounter. Mobile browser market share worldwide. https://gs.statcounter.com/browser-market-share/mobile/worldwide, 2023.

[38] Thomas Gerbet, Amrit Kumar, and Cédric Lauradoux. A privacy analysis of google and yandex safe browsing. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 347–358, 2016.

[39] Gaurav Aggarwal, Elie Bursztein, Collin Jackson, and Dan Boneh. An analysis of private browsing modes in modern browsers. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, USENIX Security'10, page 6. USENIX Association, 2010.

[40] John Korniotakis, Panagiotis Papadopoulos, and Evangelos P Markatos. Beyond black and white: Combining the benefits of regular and incognito browsing modes. In *17th International Conference on Security and Cryptography*, SECRYPT'20, 2020.

[41] A. Shuba, A. Markopoulou, and Z. Shafiq. Nomoads: Effective and efficient cross-app mobile ad-blocking. In *Proceedings of the Privacy Enhancing Technologies Symposium*, PETS'18, 2018.

[42] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374, 2016.

[43] Apostolis Zarras, Alexandros Kapravelos, Gianluca Stringhini, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, 2014.

[44] Erin Kenneally and David Dittrich. The menlo report: Ethical principles guiding information and communication technology research. *Available at SSRN 2445102*, 2012.

[45] Caitlin M. Rivers and Bryan L. Lewis. Ethical research standards in a world of big. *F1000Research*, 3, 2014.

A ETHICAL CONSIDERATIONS

The execution of this work has followed the principles and guidelines of how to perform ethical information research and use of shared measurement data [44, 45]. Hence, we keep our crawling to a minimum to ensure that we do not slow down or deteriorate the performance of the visited web services in any way, and make considerable effort not to overwhelm the hosting servers. As a result, we crawl only the landing page of each website and visit it only once. We do not interact with any component inside a website, and only passively observe network traffic. In addition to this, Panoptes has been implemented to wait for both the website to fully load and an extra period of time before visiting another website. Consequently, we emulate the behavior of a normal user that landed on a website. In accordance to the GDPR and ePrivacy regulations, we did not engage in collection of data from real users.

Revisiting IP Multicast

Sylvia Ratnasamy
Intel Research

Andrey Ermolinskiy
U.C.Berkeley

Scott Shenker
U.C.Berkeley and ICSI

ABSTRACT

This paper revisits a much explored topic in networking – the search for a simple yet fully-general multicast design. The many years of research into multicast routing have led to a generally pessimistic view that the complexity of multicast routing – and inter-domain multicast routing in particular – can only be overcome by restricting the service model (as in single-source) multicast. This paper proposes a new approach to implementing IP multicast that we hope leads to a reevaluation of this commonly held view.

Categories and Subject Descriptors: C.2.2 [Network Protocols]: Routing Protocols

General Terms: Design.

Keywords: Routing, Multicast.

1. INTRODUCTION

In 1990, Deering proposed IP multicast – an extension to the IP unicast service model for efficient multipoint communication [1]. The multicast service model offered two key benefits: (1) the efficient use of bandwidth for multipoint communication and, (2) the indirection of a group address which allows for network-level rendezvous and service discovery. Deering's proposal triggered an era of research on the implementation and applications of IP multicast. In terms of actual deployment, this research has had somewhat mixed success. On the one hand, support for multicast is built into virtually every endhost and IP router and the service is often deployed within enterprise networks. However there is little cross-provider global deployment of multicast, and today, fifteen years after Deering's seminal work, the vision of a ubiquitous multicast "dialtone" remains an elusive, if not altogether abandoned, goal.

Theories abound for why this vision was never realized (*e.g.*, [2–4]). Very broadly, most of these can be viewed as questioning the viability of IP multicast on two fronts. The first is its practical *feasibility* given the apparent complexity of deploying and managing multicast at the network layer. The second is the *desirability* of supporting multicast with many questioning whether the demand for multicast applications justified the complexity of its deployment, whether ISPs could effectively charge for the service, the adequacy of alternate solutions, and so forth.

SIGCOMM'06, September 11–15, 2006, Pisa, Italy.

This paper directly addresses the issue of feasibility, proposing a simpler approach to implementing IP multicast that we call Free Riding Multicast (FRM). We focus on inter-domain multicast for which complexity proved particularly acute but (as we describe later) FRM can be extended to the intra-domain scenario as well. FRM offers two key advantages over existing solutions:

- by leveraging existing unicast routes, FRM virtually eliminates the need for a distributed multicast route computation thus side-stepping much of the network layer complexity associated with traditional solutions (hence the name "Free Riding").
- a domain's participation and use of inter-domain multicast is effected via the same channel as in the unicast case, namely BGP, thus offering network operators a familiar framework within which to tackle the management (access control, accounting, *etc.*) of a multicast service.

These advantages however come at a cost and the core tradeoff FRM makes is to tilt the complexity of route computation to the *internals* of a router (as opposed to distributed protocol mechanism). Consequently, FRM requires more storage and algorithmic sophistication at routers and can be less efficient in bandwidth consumption than traditional multicast solutions. However this tradeoff – the avoidance of distributed computation and configuration at the cost of optimal efficiency – is one we believe is worth exploring given technology trends [5] that can endow routers with significant memory and processing on the one hand and our continued difficulties taming wide-area routing algorithms on the other [6, 7].

The primary focus of this paper is the design and evaluation of FRM. We lay the context for our work in Section 2, then discuss prior work and our overall approach in Sections 3 and 4 respectively. We present the design, evaluation and implementation of FRM in Sections 5, 6 and 7 respectively. Finally, we observe that FRM represents a more general approach to supporting network-layer services such as anycast or data-centric routing. We touch on this and other directions in Section 8.

Our contribution is a new approach to implementing multicast that we hope would lower the technical barriers to its deployment. At the same time, our exploration is triggered in part by the suspicion that the desirability of multicast too might merit rescrutiny. We briefly touch on this in the following section.

2. IN DEFENSE OF IP MULTICAST

While we make no claims to understand the "market" for multicast, we observe that many of the applications that originally motivated the research on multicast have (finally) arrived and would still be well served by native multicast support.

One example is massive multiplayer games (MMORPGs) with reports of 30-100% [8, 9] annual subscription growth and upto 5 million active subscriptions in a year [10]. In these games, a player's moves must be propagated to those in its "virtual" vicinity. Currently, game operators achieve this by deploying multiple servers, each assigned a region of the virtual world, that relay communication between players. Thus, for n nodes in a virtual region, the corresponding server's bandwidth requirements vary from O(n) to O(n^2) depending on the extent to which timeliness constraints allow multiple updates to be aggregated [11, 12]. Such scaling can be problematic and indeed numerous reports cite overloaded servers affecting the user experience [8, 12].[1] In a simple scenario, game operators might use multicast to cut server bandwidth to between O(1) to O(n). In a more sophisticated scenario, players might directly multicast updates thus offloading data forwarding from servers. In short, IP Multicast can aid game operators in building more lightweight, and hence ultimately cheaper infrastructure.

Another example is the adoption of Internet TV technology [15] with several providers already in customer trials. These efforts use IP multicast within their networks but currently rely on pre-provisioned channels from the content source in to their networks. Such provisioning allows ISPs to deliver content from major content providers to their immediate access customers. Supporting multicast across domains would further allow ISPs to transitively extend this delivery to more viewers and content providers without requiring each content provider to partner with individual ISPs.

File-sharing, software updates, RSS dissemination, video conferencing, grids are additional examples of deployed services that could potentially leverage multicast delivery.

It has been argued however that it is difficult for ISPs to charge for the use of multicast, leaving them with little incentive for deployment. As Diot *et al.* observe [3], this has much to do with the open, available-to-all usage model of earlier research. We assume that ISPs will instead enforce a more closed access model enabling them to charge for and better control usage (Section 8). Given a closed usage model, the emergence of ISP hosting services and IPTV lend hope that viable charging models exist; *i.e.*, ISPs charge server operators and/or endusers for multicast connectivity.

A rejoinder is that alternate techniques such as source-specific (SSM) or application-layer multicast can meet the needs of the above applications. Regarding SSM, we note that multi-source applications do exist (*e.g.*, game servers exchanging summary updates, P2P, video conferencing) and FRM could support these with complexity comparable (although different in nature) to SSM. SSM also loses the rendezvous features of the more general service model; while difficult to assess precisely, the many uses of a DHT's indirection capabilities [16–18] and the interest in auto-discovery mechanisms that enable opportunistic [19, 20] or configuration-free networking [17] suggest that low-level rendezvous might be a broadly useful feature.

Which leaves us with application-layer solutions. While both network and application layer solutions offer the benefits of multicast – efficient multipoint communication and indirection – they do so with very different tradeoffs. While application layer solutions are less constrained by the operational concerns of ISPs, scaling these to a global user population with even modest per-user bandwidth requirements represents a serious investment in bandwidth, server resources, and management. For example, the global adoption of IPTV with no support for multicast would require the ubiquitous deployment of video servers down at the DSLAM level. Moreover, deployments of such scale are likely beyond the resources of any single application provider but independent deployments bring with them non-trivial issues of application-level peering and interoperability. Network-layer solutions by contrast, allow the deployment of services that scale by augmenting an existing global ecosystem of infrastructure, services, customers, and peering arrangements. As a deployment vehicle, this can be particularly attractive for general-purpose services such as multicast or rendezvous, that can serve a variety of applications. The clear downside is that any evolution to this complex ecosystem is inevitably constrained.

While these tradeoffs are well-recognized, the reputed complexity of IP multicast has had the unfortunate consequence of transforming the debate on the desirability of IP multicast into one of whether it is strictly *necessary* to support multicast in routers. By lowering the complexity of network-layer multicast, we hope instead to revert back to debating its utility. In this context, the above discussion offers examples of existing applications that stand to gain from ISP deployment of IP multicast. We conjecture that ultimately both network and application-layer solutions to multicast – each used as appropriate – have much to offer in the implementation and scaling of networked applications such as network games, IPTV, *etc.* and it would be valuable to leave the door open to both approaches.

3. FRM: BACKGROUND, APPROACH

IP Multicast offers endhosts a simple abstraction: a host can *join* or *leave* a multicast group G and any host can *send* to a group G. As with unicast, the internals of the network provide the foundational packet delivery service atop which richer functionality may be implemented at endsystems. The multicast routing problem is thus key to supporting the service model and has been the subject of much research over the years [4, 21–27]. We start with a brief review of this literature. In the following, we distinguish between multicast routing and forwarding – the former to refer to the construction of distribution trees, the latter to the process by which routers propagate packets.

3.1 Multicast Routing

Deering's early work defined DVMRP, a broadcast-and-prune approach in which a packet multicast by a source S is forwarded towards *all* endhosts and those that receive unwanted packets send "prune" messages up the distribution tree toward the source [1, 21]. DVMRP constructs efficient shortest-path trees from *any* source but scales poorly for which reason it is typically limited to intra-domain routing.

Another intra-domain protocol is MOSPF, a multicast extension to unicast OSPF in which a router augments its link state advertisement with the list of groups for which it has local receivers which allows all routers to compute the shortest path tree from *any* source to all receivers. MOSPF is a fairly incremental extension and builds efficient trees but is limited to networks that run link-state protocols.

Shared tree protocols such as Core-Based Trees (CBT) [22] and PIM-SM [23] emerged to address the poor scaling of flood-and-prune tree construction. These protocols associate a special rendezvous point (RP) router that serves as the root of a single tree shared across all senders for a group. A new receiver sends a JOIN message along the unicast path towards the group's RP, instantiating forwarding state at routers along the way. While shared-tree protocols offer a dramatic improvement in scalability, they give rise to non-trivial issues regarding the placement and discovery of RPs.

[1] A revealing anecdote is the virtual demonstration in which, to protest issues with Warcraft's [13] operators, players overloaded and crashed operator servers by assembling in one virtual location [14].

Perhaps more importantly, the RP is the nerve center that determines the very availability of a PIM-SM tree and hence ISPs proved reluctant to depend on RPs run by other ISPs. This led to the development of the Multicast Source Discovery Protocol [24] that allows domains to discover and interconnect multiple RPs in a loose mesh. To accommodate the incremental deployment of PIM-SM/MSDP, multi-protocol extensions were introduced in BGP-4 (MBGP) [27]. MSDP has its own scaling problems and was thus originally intended as a temporary measure pending the deployment of a more scalable inter-domain solution.

BGMP [25] is one such proposal and incorporates many of the above ideas [25]. BGMP supports source-rooted, shared and bidirectional shared trees. Key to BGMP is the association of a group to a "home" AS responsible for allocating the group address. A group's home AS acts as the domain-level RP for the group's routing tree. To map a group to its home AS, BGMP proposes address advertisement (AAP) [25] that may be used in conjunction with MASC [28], a dynamic address allocation protocol.

The ever increasing complexity of multicast routing led Holbrook *et al.* [4] to challenge the wisdom of Deering's service model. They argued that many large-scale applications only require delivery from a single, often well-known, source. By exposing the identity of this source to the endpoints, routing can be greatly simplified by having receivers just send JOIN messages directly towards the source, moving RP discovery out of routers. Their Express protocol (now PIM-SSM) thus proposes a *single-source* service model in which a multicast "channel" is identified by both a group (G) and source (S) IP address. Endhost `joins/leaves` specify an (S,G) channel address and only the source S may transmit to a channel.

Holbrook *et al*'s insight represents a practical compromise that has done much to further ISP adoption of IP Multicast. The price is a loss in generality – with SSM, a group address is tied to a specific endhost IP address and hence the value of multicast as a network-layer rendezvous mechanism is largely lost. FRM makes a different compromise – retaining generality and seeking simplicity by accepting higher bandwidth and (off-the-fast-path) storage costs at routers.

3.2 Multicast Forwarding

The above centered on efforts to scale multicast routing. Of at least equal concern is the scalability of multicast forwarding state within routers. Because group membership need not be topologically contained, multicast forwarding entries are not easily aggregatable and, left unchecked, forwarding state grows linearly in the number of groups that pass through a router. Thaler and Handley [29] propose an interface-centric implementation model applicable to shared-bus router architectures which allows some aggregation. Their implementation model however does not apply to switched router architectures nor implementations which store forwarding state as a list of per-group incoming-outgoing interfaces. Moreover, in the absence of careful address allocation, forwarding state remains fundamentally linear in the number of active groups and can hence be non-trivial. Radoslavov [30] proposes "leaky" aggregation that tradesoff bandwidth for scalability in state while Briscoe *et al.* [31] propose a scheme wherein applications cooperate to select addresses that aid aggregation. To the best of our knowledge, none of these schemes have been adopted in common router implementations.

Discussion. The quest for a satisfactory multicast routing solution thus led down an increasingly tortuous path. Perhaps reflective of this is the somewhat daunting list of multicast protocols found in most commercial routers; *e.g.*, Cisco routers advertise implementations of PIM-SM, PIM-DM, Bidir-PIM, PIM-SSM, AutoRP, MBGP, MSDP and IGMP v1,v2,v3 (while still lacking support for address allocation (MASC/AAP) and scalable inter-domain (BGMP) protocols!) It is hard, even in retrospect, to cleanly lay the blame for this abundance of mechanism at the feet of any one problem or issue as each solution addresses a very real concern. Unfortunately, the complexity of the ensemble greatly raises the barrier to deployment of a multicast service. Our primary goal with FRM was thus to provide a "leaner" solution while retaining acceptable performance. Next, we describe our approach to achieving this.

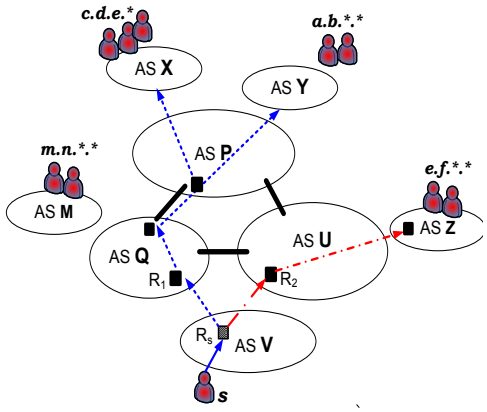


Figure 1: ***FRM: group membership and forwarding.***

4. FRM: APPROACH AND TRADEOFFS

4.1 Approach

In the abstract, multicast delivery requires knowledge of: (1) which end hosts are group members and, (2) how to reach these member hosts or domains. While most solutions combine these two components into a single from-the-ground-up protocol, FRM decouples membership discovery from route discovery. This separation offers the advantage that, once group members are known, any source can construct the multicast tree from its unicast routes to each member host. This is easily done for path-vector or link-state unicast protocols that reveal the required paths; as we discuss in Section 8, this approach can also be adapted to distance-vector protocols.

As stated earlier, we focus on inter-domain routing in which scenario the basic FRM scheme operates as follows: a domain's BGP advertisements are augmented with a description of the multicast groups currently in use within the domain. These adverts are then propagated as per the normal operation of BGP thereby giving every (border) router a description of the groups present in each destination prefix. To discover the dissemination tree for a group G, the the border router at the source (denoted R_s) scans its BGP table to identify those prefixes with members of G. Having thus identified all destination domains, R_s simply computes the dissemination tree from the union of the BGP unicast paths to all destination domains. R_s then forwards a single copy of the packet to each next hop on this dissemination tree along with an encoding of the subtree each next hop must in turn forward the packet along.

Figure 1 illustrates this process: A packet multicast to G by host s arrives at R_s. From their BGP advertisements, R_s learns that prefixes $a.b.*.*$, $c.d.e.*$, and $e.f.*.*$ have members in G and computes the multicast tree from the BGP paths from V to each of the

above prefixes and forwards one copy of the packet to Q along with an encoding of the subtree in dashed-line and another copy to U with an encoding of the subtree in dash-dot-dash style.

4.2 Discussion

To some extent, FRM can be viewed as extending MOSPF to the inter-domain arena. This extension however is non-trivial because we do not have a complete network map at the inter-domain level. The path-vector nature of BGP allows a router to compute the shortest path(s) from *itself* to a set of receivers but not from *any* source to a set of receivers. This complicates forwarding as a router that receives a packet has no way of knowing which subset of receivers it should forward towards since it does not know whether it lies on the shortest path from the source to those receivers. For example, in Figure 1: R1 and R2 both have BGP entries for prefixes $c.d.e.*$, $a.b.*.*$, and $e.f.*.*$, and can hence infer the presence of group members in these prefixes. However, when R1 receives a packet from R_s, it has no easy way of knowing not to forward toward $e.f.*.*$ and likewise R2 towards $c.d.e.*$ and $a.b.*.*$. While one might employ a limited form of flood-and-prune this raises issues similar to DVMRP in terms of scalability and vulnerability to dynamics. An alternate option might be to change BGP to a policy compliant link-state protocol however this represents a major overhaul of BGP which we avoid. FRM's forwarding is instead designed to exploit and live within the constraints of the information BGP offers. Finally, we note that while PIM and Express too leverage existing unicast routes they do so only in forwarding JOIN messages towards the rendezvous point; packet delivery still relies on group-specific forwarding state laid down by JOINs.

4.3 Tradeoffs

The core tradeoff FRM makes is to cut down on distributed protocol mechanism at the cost of demanding more from the *internal* capabilities of routers. This offers both advantages and challenges. On the positive side, we offer the following observations:

Parsimony in protocol mechanism. In terms of protocol complexity the basic FRM framework requires: (1) extending BGP to carry group membership information and (2) that an AS on occasion filter some group for a downstream customer AS (for reasons described in Section 5).

ISP control. Because group membership is explicitly advertised through BGP, an ISP has ultimate (and easy) control over which groups its customers subscribe to; *e.g.*, to block an undesired group, an ISP can simply drop it from its BGP advertisement. FRM also allows ISP control over sources in its domain as border routers have knowledge of (and control over!) the destination domains included in the dissemination tree. As articulated by Holbrook *et al.*, this assists in source-based charging as an ISP can now infer the traffic "amplification" due to a multicast transmission.

Ease of configuration. FRM avoids the contentious selection of RPs and new inter-domain protocols, instead piggybacking membership state over BGP.

Centralized route construction. In FRM, the multicast tree is computed in its entirety by the source's border router using existing unicast routes. This not only eliminates the need for a separate multicast routing algorithm but also spares us new routing anomalies [6, 7].

General service model. FRM supports a multi-source service model with efficient source-rooted trees.

The key challenges FRM faces include:

State requirements. FRM incurs the overhead of advertising and maintaining group membership. While true for all multicast protocols, FRM disseminates membership information more widely than traditional protocols and hence incurs greater overhead. Specifically, group state in FRM is aggregated per destination prefix rather than on the basis of topology.

Unorthodox packet forwarding. Traditional packet forwarding involves a (longest prefix match) lookup on the destination address to obtain the next hop along which to send the packet. By contrast, in FRM, finding the next hop(s) requires that the access border router scan its entire BGP table and that intermediate nodes decipher the encoded tree. FRM faces the challenge of achieving this in a manner that is both scalable and amenable to high-speed forwarding.

Bandwidth overhead. FRM's use of what is effectively a form of multicast source routing incurs additional bandwidth costs.

The remainder of this paper presents the design and evaluation of a protocol that addresses the above concerns.

5. DESIGN

The design of FRM comprises two (mostly separable) components – group membership discovery and multicast packet forwarding. This section presents our solutions for each along with a qualitative evaluation of their resource requirements. Our design assumes the license to quite significantly modify a router's internal operation though we do not modify unicast processing and require only a modest (and hardware-friendly) upgrade to forwarding plane. This appears reasonable given vendors' past willingness to incorporate new multicast routing into their routers. Our discussion of the overhead due to packet processing in routers follows standard assumptions – that high speed forwarding is assisted if packets are processed entirely on line cards and that the memory and processing available at the route processor may be comparable to high end machines but is more limited at line cards.

5.1 Advertising Group Membership

To leverage unicast routes, group membership information must be maintained at the same granularity as unicast routing destinations. For this, FRM augments BGP to include per-prefix group membership information. A border router augments its BGP advertisements with a description of the group addresses active – *i.e.*, with at least one member host – within its domain. Because simple enumeration leaves little opportunity for scaling to large numbers of groups, we encode active group addresses using bloom filters which allow advertisements to be compressed in a manner that introduces false postives but no false negatives and hence never results in service being denied to valid group members. The possibility of false positives however implies that a domain may on occasion receive traffic for a group it has no interest in. To handle this, the receiving domain R can either simply drop the unwanted traffic or, similar to DVMRP, can inform the upstream domain U to cease forwarding traffic for that particular group. This latter can be implemented by installing an explicit filter rule at U or by having R recode its advertisement to U into multiple bloom filters such that the offending false positive is eliminated. In this paper, we assume the use of filter rules.

Through the intra-domain multicast protocol (Section 5.3), a border router discovers which groups are active in its local domain and

encodes these addresses into a group bloom filter, denoted GRP_BF. The length of a GRP_BF is selected by reasoning in terms of the number of filter entries an AS is allowed by its upstream ASes. Each false positive results in a filter being installed at the upstream provider's network and hence, if an AS is allowed f upstream filters, then we set its target false positive rate to $\text{MIN}(1.0, f/(A-G))$ where G is the number of groups to be encoded and A is the total size of the multicast address space. This choice follows from the observation that a false positive can only be triggered by one of $A-G$ addresses which improves scalability by allowing for appropriately smaller GRP_BFs at large G; *e.g.*, a domain with $G \sim A$ ought only use a single bit that tells upstream domains to forward all multicast traffic its way. The filter size L is then computed using the above false positive rate. For efficient manipulation (compression, aggregation, expansion), we require that L be a power of two and assume a well known maximum length L_{max}.

A border router then piggybacks GRP_BFs on its regular BGP advertisements. If customer prefixes are aggregated, a corresponding aggregate GRP_BF is computed as the bitwise-OR of the individual customer GRP_BFs . Finally, based on its available memory, a router can independently choose to compress a GRP_BF of length L by repeated halving wherein the filter is split in two halves that are then merged by a bitwise-OR. Inversely, a previously compressed bloom filter can be expanded by repeated concatenation to obtain the desired length. Of course, both aggregation and compression result in a corresponding increase in the false positive rate.

Memory requirements. The total memory due to GRP_BF state at a participant border router is on the order of the number of destination prefixes times the average GRP_BF length. This can be non-trivial – for example, our evaluation in Section 6 estimates GRP_BF memory for 170,000 prefixes and 1 million active groups at approximately 2 GB. Fortunately, FRM's forwarding scheme does not require that GRP_BF state be stored in the forwarding tables on individual line cards and instead places GRP_BF state in the BGP RIB on the route processor. As such, the main impact due to the memory requirements for GRP_BF state is the monetary cost of memory. At even current memory prices, this should to be a minor increment to overall router costs [5].

Bandwidth and processing requirements. In keeping with the incremental nature of BGP, changes in GRP_BFs are communicated as deltas and hence the rate of updates depends primarily on the rate at which groups are added to, or removed from, a GRP_BF. Advertisements are for the domain as a whole and hence require updating only when the number of group members drops below one or rises above zero and hence unlikely to fluctuate rapidly, particularly if withdrawals are damped (as is likely [32]). Moreover, updates are small – on the order of the number of bloom filter hash functions for each added/deleted group. In terms of processing, GRP_BF updates, unlike BGP route updates, do not trigger route recomputations and only rarely require updating the actual forwarding tables on line cards (we describe when this is needed in the following section). Instead, processing GRP_BF updates is largely a matter of updating the BGP RIB in the route processor's memory. Thus, both the frequency and processing overhead due to GRP_BF updates should be tractable.

5.2 Multicast Forwarding

FRM processes packets differently at the border router in the access domain for the source (R_s), and border routers in the transit core (R_t). We discuss each in turn.

Forwarding on GRP_BF *state at* R_s. A packet multicast by source s to a group G is delivered via the intra-domain multicast routing protocol to R_s, the border router in the source's domain. R_s scans its BGP RIB, testing each GRP_BF entry to identify the destination prefixes with members in G and constructs the AS-level multicast tree $T(G)$ from the union of the individual AS-level paths to each member prefix. $T(G)$ can be computed in $O(p \times d)$ where p is the number of prefixes and d the average AS path length. We assume these operations are performed by the route processor where GRP_BF state is stored. While rather expensive, two factors render this computational complexity manageable. First is simply that, as an access router, R_s is under less forwarding load (in terms of both number of groups and total packets) than core routers and is hence better positioned to absorb this overhead. Second, and more valuable, is that R_s can cache, or even precompute, the results of the lookup so that this computation is only invoked on the first packet sent to each group. Thus, the complexity of lookups on GRP_BF state is incurred only by access border routers and, even there, only once for each group with active sources in the local domain.

Forwarding on cached state at R_s. As described above, R_s caches the results of the initial lookup on a group address G. Cached forwarding state is indexed by group address and hence accessed by exact-match lookups. Many well-known techniques exist for efficient exact-match lookups and we assume that FRM would employ any of these as appropriate – *e.g.*, CAMs and direct-memory data structures offer O(1) exact-match lookups while more compact data structures achieve exact-match lookups in logarithmic time [33, 34]. The total memory requirements for cached forwarding state depends on the number of groups with active sources within the domain and the per-group forwarding state. The latter of these depends on the size of the tree T(G) (we enumerate the exact forwarding state R_s must cache in the discussion on forwarding at R_t that follows). Our evaluation in Section 6 suggests that this state could be mostly accommodated in RAM on line cards – for example, our evaluation estimates a 400MB cache for a domain that has simultaneously active sources for 1 million groups [35–37]. If the memory on line cards cannot accommodate the entire cache, one might only cache state for high data rate groups on line cards leaving the route processor to handle forwarding for low data rate groups. Our implementation achieves this with LRU cache replacement.

In summary, caching replaces the linear scan of the BGP RIB's GRP_BF state by an exact-match lookup on cached forwarding state and, if needed, should be mostly achievable in line cards. We note that R_s maintains per-group forwarding state. However, as mentioned earlier, we believe this scaling is reasonable here because the number of groups (with active sources) in R_s's domain is likely lower than in core transit domains. In fact, the intra-domain multicast protocol is likely to impose similar scaling.

Forwarding at R_t. Multicast delivery is now a matter of forwarding the packet along $T(G)$, the AS-level tree computed by R_s, with appropriate packet replication at fanout domains. However, as described in Section 4, R_s cannot simply forward the packet to each of its next hop ASes on the tree as an interior AS does not know which subset of destination prefixes it should in turn forward to. Moreover, such an approach would impose forwarding state and complexity akin to that at R_s on all routers – a scenario we've argued against. We instead adopt an approach in which R_s communicates T(G) to intermediate routers. FRM implements this using a "shim" header above the IP header into which R_s encodes the *edges* from $T(G)$. A tree edge from autonomous system A to B is

State	scaling	lookup	used at	stored in	when used
GRP_BFs	O($\|p\|.g$)	linear scan	R_s	route proc.	per group
cached GRP_BFs	O($g_s.T(g_s)$)	exact match	R_s	line card	per pkt
encoded links	AS degree	filter match	R_t	line card	per pkt

Table 1: ***FRM: packet processing requirements.*** $|p|$ ***is the total number of prefixes at a BGP router and*** g ***the average groups per prefix.*** g_s ***is the number of groups with active sources in domain*** s ***and*** $T(g_s)$ ***the average size of the dissemination trees for groups with source in*** s***.***

assigned the unique label 'A:B' and R_s encodes these edge labels into the shim header it constructs for each of its next hops. Hence, in Figure 1, R_s would encode 'Q:P', 'P:X' and 'P:Y' in its packets to $R1$ and 'U:Z' in those to $R2$. Note that our choice of encoding edge labels is actually crucial in allowing R_s to disambiguate forwarding responsibility amongst interior ASes and allows the shim header inserted at R_s to be carried unchanged all the way to the destination(s) with no updating at intermediate routers (*e.g.*, this would not be possible were R_s to encode only the AS numbers of *nodes* in the tree).

For scalability reasons similar to those discussed in Section 5.1, we encode the dissemination tree into the shim header using a bloom filter (denoted TREE_BF) and deal with false positives as described in Section 5.1. However, unlike the GRP_BF advertisements, we require that the TREE_BF be of fixed length – or one of a small set of well-known lengths – so as to be amenable to fast processing in hardware. This raises the issue of picking an appropriate TREE_BF length. A too small header can lead to high false positive rates for large groups while a TREE_BF length selected to accommodate even the largest groups would be needlessly wasteful in the per-packet overhead the shim header imposes. Our solution instead is the following: we pick a fixed TREE_BF size of h bits, a target false positive rate f and compute e, the number of edges that can be encoded in h bits while maintaining a false positive rate $\leq f$. We then use a standard bin-packing algorithm to decompose the tree into groups of subtrees such that the number of edges in each group is less than e. This bin-packing can be computed in a single (typically partial) pass over $T(G)$. Each group of subtrees is then encoded into a single shim header and transmitted as a separate packet. Note that this approach can cause certain links to see multiple copies of a single multicast transmission.

The "tax" due to our *source-encoded* forwarding is thus twofold (we quantify these in Section 6):

- in its bandwidth consumption, source-encoded forwarding can be more inefficient than traditional multicast due to the per-packet shim header and redundant transmissions (on certain links) for groups too large to be encoded into a single shim header.
- the per-group forwarding state cached at R_s must now include the shim header(s). *I.e.*, for each cached group G, R_s caches the list of next hop ASes and the shim header(s) associated with each next hop.

The payoff is highly scalable and efficient packet processing at intermediate routers – to forward a packet, R_t need only check which of its AS neighbor edges are encoded in the shim header's TREE_BF. *I.e.*, if A is R_t's AS number, then, for each neighbor AS B, R_t checks whether 'A:B' is encoded in the packet's shim header in which case it forwards a copy of the packet to B. This offers two advantages. The first is that R_t's "forwarding" state is essentially a list of its neighbor edges. This state is independent of any particular group G and hence the number of such forwarding entries at R_t depends only on its domain's AS degree. Measurements report per-domain AS degree distributions ranging from 1 to under 10,000 with a power-law distribution and hence we can expect the number of forwarding entries at R_t to be low– potentially several orders of magnitude lower than the number of multicast groups – and easily accommodated on line cards.

For efficient packet processing, we store neighbor edges in their encoded representation; *i.e.*, each edge is inserted into, and stored as, a separate TREE_BF bloom filter. The lookup operation at R_t is then similar to standard filter matching – for each neighbor edge, R_t checks whether the corresponding bits are set in the packet's TREE_BF. There are a variety of options by which to implement this but perhaps the simplest is to use TCAM with the bloom filter for each neighbor edge stored in one TCAM row and all zero bits set to the "don't care" value [40]. With this, all edges can be matched in parallel with a single TCAM access. Alternately, this state can be stored in RAM and an edge matched in logarithmic time. Finally, as mentioned before, the shim header remains unmodified along the entire path and requires no updating at R_t.

The second advantage to source-encoded forwarding is that, because the forwarding state at R_t depends only on its (mostly static) set of AS neighbors, no wide-area protocol mechanism is required to construct and maintain this state. Source-encoded forwarding thus achieves sparseness in protocol mechanism and scalable forwarding state though at the cost of some additional bandwidth and memory (at R_s) usage.

Table 1 summarizes the state requirements due to FRM. We note that while FRM tilts the burden of forwarding state and complexity onto source access domains, this is a not displeasing arrangement as the benefit of multicasting is greatest at the source (a receiver's bandwidth consumption is unchanged with multicast). Finally, we note that source-encoded forwarding (somewhat unlike IP source routing) is easily implemented in hardware and selects paths compliant with the policy choices of intermediate ISPs.

5.3 FRM and Intra-domain Protocols

Many of the operational and scaling issues that complicate inter-domain multicast routing are less acute in the intra-domain scenario and hence it appears reasonable to retain existing solutions (*e.g.*, PIM-SM, DVMRP) at the intra-domain level. These can interface to FRM in a straightforward manner; *e.g.*, a group's internal RP could notify border routers of domain-wide group membership and packets could be relayed to/from FRM border routers via tunneling to the RP or by having border routers join groups with active sources. If desired however, FRM could be extended to the intra-domain scenario; we briefly discuss this in Section 8.

6. EVALUATION

In this section, we use simulation and trace-driven calculation to estimate the storage and bandwidth overhead due to FRM's group membership and forwarding components. Due to space constraints, we present only key results for likely usage scenarios. a more detailed parameter exploration is presented in [41].

Setup. To relate performance directly to end-user behavior, we allow U=2^{32-p} users in a domain of prefix length p and assume that each user joins k groups selected using some group popularity distribution from a total of A *simultaneously* active groups. Unless stated otherwise, we model group popularity using a zipfian distri-

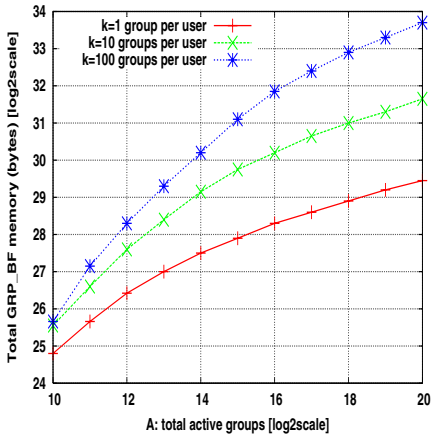


Figure 2: ***Total GRP-BF storage per border router.***

bution akin to the Web [42] and pessimistically assume no locality in group membership; any locality would only improve scalability. We use Subramanian *et al.*'s Oct'04 snapshots of BGP routing tables and their AS-level topologies annotated with inter-AS peering relationships [43].

6.1 Group Membership

Memory overhead. Per-prefix GRP_BFs are required to store group membership information. We compute the GRP_BF size for a single prefix as follows: the number of groups advertised per prefix – denoted G – is the expected number of distinct groups given that U users each pick k-from-A as per the selected group popularity distribution and hence the corresponding GRP_BF size is the bloom filter size needed to encode G items for a target false positive rate of $f/(A-G)$ (recall that f is the target number of filters per prefix). Then, to estimate the *total* storage due to GRP_BF state at a BGP router, we use real BGP tables [43] and compute the total storage per router as the sum of the GRP_BF size corresponding to each prefix entry. Figure 2 plots this total storage for increasing A for f =10 and k =1, 10, and 100 groups per user.

Overall, we see that the memory required to maintain group membership state, while not trivial, is very manageable given current storage technology and costs. For example, 1 million simultaneously active groups and 10 groups per user requires approximately 3 GB – an amount of memory found today on even user machines. Moreover, the trend in memory costs should allow FRM to handle the relatively slower growth in BGP table size.

Bandwidth costs. We use back-of-the-envelope calculations to show that the bandwidth due to updating group membership is tractable. Recall that a domain updates its membership for group G only when the number of members of G within the domain falls to, or rises above, zero. Moreover, some domain-level damping of group departures is likely. We thus generously assume a prefix sees a new group appear or an existing group depart every second. Updates are conveyed as the set of GRP_BF bit positions to be set/reset. Hence if we assume GRP_BFs use 5 hash functions and bit positions are represented as 24 bit values (in multiples of 256-bytes), then updating membership for a single prefix requires approximately 15 bytes per second (Bps). If we assume a router with full BGP routes has 200,000 prefix entries (current reports indicate ∼170,000 FIB entries [44]) then the total bandwidth consumed due to updates would be approximately 3MBps – a small fraction of the bandwidth capacity at core BGP routers.

The first node to join a group within its prefix/domain incurs the latency due to inter-domain GRP_BF update propagation. (The latency of subsequent joins is that of an *intra*-domain join.) Unlike regular BGP updates, GRP_BF updates do not trigger distributed route recomputations and hence their rate of propagation will likely be limited primarily by protocol constraints (if any) used to bound update traffic (as opposed to concerns about routing loops, inconsistencies, and the like). Our current prototype limits inter-AS GRP_BF updates to once per second which would lead to a "first-time" join latency of ∼ 1-6 seconds given current AS path lengths [44]. Further deployment experience would be required to better gauge appropriate update intervals.

Group size	Ideal multicast	FRM	per-AS unicast
100	28	28	38
1000	158	159	246
10,000	1000	1012	1962
100,000	4151	4233	9570
1M	8957	9155	21754
10M	15353	15729	39229

Table 2: `total-tx`***: the total number of packet transmissions for increasing group sizes.***

6.2 Forwarding Overhead

Bandwidth costs. The bandwidth overhead due to FRM forwarding stems from: (1) the per-packet shim header and, (2) the redundant transmissions required when subtrees are too large to be encoded in a single shim header. We assume fixed 100 byte shim headers and measure the overhead in packets transmitted; our results extrapolate to different shim header sizes in a straightforward manner.[2]

We use two metrics to quantify FRM's overhead due to redundant transmissions:

- `total-tx`: the total number of packet transmissions required to multicast a single packet from the source to all receivers
- `per-link-tx`: the number of transmissions *p*er link used to multicast a single packet from source to all receivers.

To calibrate FRM's performance, we measure the above for: (1) "ideal" multicast in which exactly one packet is transmitted along each edge of the source-rooted tree and, (2) per-AS unicast in which the source unicasts each member AS individually. This latter can be achieved using only FRM's group membership component and thus represents a simple network layer solution that requires no multicast-specific forwarding at routers (as does FRM).

Table 2 lists `total-tx` for increasing group sizes. We see that for all group sizes, the overall bandwidth consumed by FRM is very close to that of ideal multicast (between 0-2.4% higher) while per-AS unicasts can require more than twice the bandwidth of ideal multicast. As expected, the difference between FRM and ideal multicast grows with increasing group size due to the multiple shim headers needed to encode the larger trees.

[2]100 bytes represents ∼10% overhead on typical data (*i.e.*, non-ack) packets which appears reasonable. In practice, for greater efficiency, a source might choose from a few well-known shim header sizes; *e.g.*, we find even 20B headers would suffice for groups of upto a few thousand.

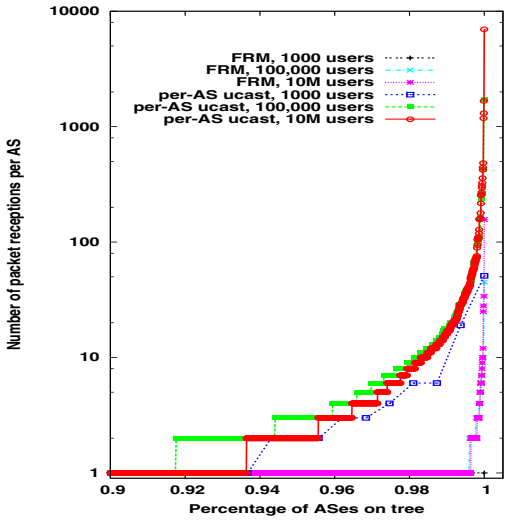


Figure 3: ***CDF of*** `per-link-tx`***, the transmissions per AS link for FRM and per-AS unicasts.***

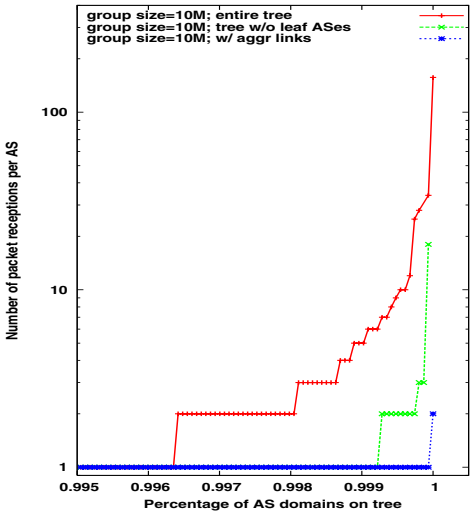


Figure 4: ***CDF of transmissions per (AS) link with optimizations to reduce the size of the encoded tree.***

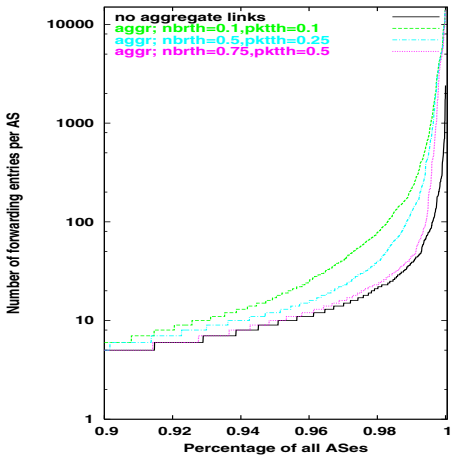


Figure 5: ***CDF of forwarding entries per AS. Tests with aggregate links use a group size of 10 million.***

Figure 3 plots the CDF of `per-link-tx` for FRM and per-AS unicasts for different group sizes (`per-link-tx` is always one for ideal multicast). In all cases, over 90% of links see exactly one transmission per link. However, we see that with per-AS unicasts, the worst-case `per-link-tx` can be over 40 for group sizes of just 1,000 and almost four orders of magnitude greater than ideal multicast for very large group sizes. FRM's tree-encoded forwarding significantly reduces this overhead as over 99.5% of links see exactly one transmission and the worst-case `per-link-tx` (at 10M users) drops to 157 relative to 6950 transmissions for per-AS unicasts.

We note that this is a stressful scenario – for our trace, 10 million users selected with no topologically locality results in every AS having a group member and is thus equivalent to broadcasting to the *entire* Internet. In such cases, FRM's overhead of $\sim$ 150 transmissions on a single link might well represent a reasonable penalty. Nonetheless, we look for techniques to further reduce this overhead. Examination reveals that the highest `per-link-tx` occurs at large ISPs that have both high degree and a large number of downstream ASes (*e.g.*, ATT, Usenet, Level-3). This leads us to propose two performance optimizations – one fairly trivial and another that, while light on mechanism, requires more forwarding state at core routers.

Optimization#1: no leaves. Here, customer ASes at the leaves of the dissemination tree are not encoded into the shim header. This could be acceptable because a provider AS that receives traffic for a group G can easily determine which of its immediate customer ASes have advertised membership in G and forward traffic appropriately. Now however, a multi-homed customer AS may on occasion receive traffic from more than one upstream provider. In this case the customer AS can, as in the event of a false positive, push filter requests to the provider sending it unwanted traffic. From figure 4, we see that this improves the worst-case transmissions per link by approximately an order of magnitude.

Optimization#2: aggregate links. If the number of tree edges from an AS A is a large fraction of either A's total edges (*nbrthresh*) or the total edges per packet (*pktthresh*), then the encoding router R_s replaces the edges from A by an *aggregate* edge 'A:$*$' that tells A to forward the received packet on all outgoing edges. Figure 4 plots the transmissions per link for *nbrthresh* $=$ *pktthresh* $= 0.5$ while Table 3 reports the worst-case transmissions per links for different *nbrthresh* and *pktthresh*.[3] We see that the use of aggregate links can allow FRM to match optimal multicast.

Aggregate links implicitly include non-tree edges. To avoid A sending packets out along non-tree edges, when A receives a packet matching 'A:$*$', it forwards the packet to a neighbor B only if the packet also matches 'B:X' for some X, neighbor of B. This requires that A know of B's edges that lie on the path from A to various destinations. Fortunately, this information is locally available from A's

[3]We note that the parameters (*nbrthresh* and *pktthresh*) do not require to be globally consistent and are instead selected independently by R_s. Moreover, the effectiveness of a particular parameter choice is immediately evident when decomposing the tree and hence R_s can experiment with a few parameter choices to achieve a target overhead.

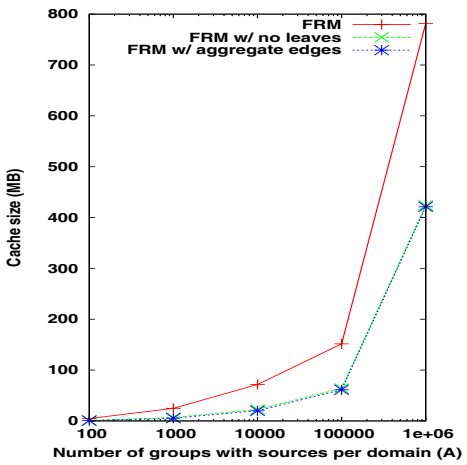


Figure 6: ***Cache size for increasing A, the number of groups with active sources in a domain.***

BGP table and can hence be obtained with no additional protocol mechanism but requires that A store additional AS edges in its forwarding table. To control this increase, A can maintain 2-hop edges for only a few neighbors and indicate these through (for example) a flag associated with a BGP path it advertises. In our tests, we assume an AS maintains 2-hop edges for only its customers and measure the corresponding increase in forwarding state.

In summary, for very large groups, aggregate edges can improve the efficiency FRM to match optimal multicast at the cost of additional forwarding state but little new mechanism (specifically, an additional BGP flag attribute, a new conditional clause in the tree decomposition at R_s and an additional matching rule in the forwarding at transit R_t routers).

Storage costs. The forwarding state at a core router R_t is made up of its AS neighbor edges and hence the number of forwarding entries at R_t is the AS degree of its domain. The use of aggregate links adds additional 2-hop edges to the forwarding table. Figure 5 plots the cummulative distribution of the number of forwarding entries per AS for both basic FRM, and FRM using aggregate edges. We see that the power-law AS degree distributions means that the vast majority of ASes have remarkably small forwarding tables – in all cases, over 90% have less than 10 entries. We also see that for most ASes the number of forwarding entries is unchanged by the use of aggregate edges. The worst-case number of entries however increases from approximately 2,400 without aggregate links to 14,071 with aggregate links. While a significant relative increase, this is still a small number of forwarding entries in the absolute. The corresponding memory requirements can be computed as the number of entries times the size of the bloom filter (recall we store each edge as a bloom filter). With 100 byte bloom filters, this gives a worst-case forwarding table of 2,400 entries, $\sim$ 240KB for FRM and 14,071 entries, 1.4MB for FRM with aggregate edges both of which can be comfortably accommodated with current TCAM usage [37, 45].

The forwarding state at the source's border router R_s consists of the cached shim header(s) for those groups with active sources within the domain. To compute the amount of cached state, we assign a domain a total of A groups with active sources and assume, as before, that users join each group based on a zipfian group popularity distribution and enforce a minimum group size (of all 8 domains) to avoid empty groups. For each resultant group size, we

nbrthresh $\Rightarrow$ pktthresh $\Downarrow$	0.1	0.25	0.5	0.75
0.1	1	1	2	2
0.25	1	2	2	3
0.5	1	2	2	6

Table 3: ***Worst-case transmissions per (AS) link with aggregate links and different*** *nbrthresh* ***and*** *pktthresh.*

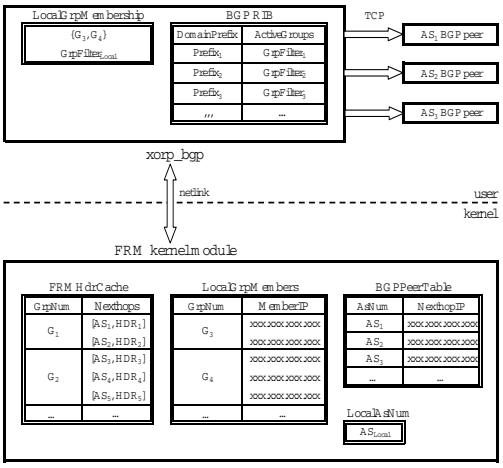


Figure 7: ***Software architecture of the FRM prototype***

compute the corresponding number of shim headers as above. Figure 6 plots the cache size for increasing A. If we assume on the order of several hundred megabytes of RAM on line cards, then we see that R_s could support line-card-only forwarding for upto several hundred thousand groups and over a million groups using the above optimizations. The initial sub-linear scaling trend is because cache requirements for highly popular groups dominate the initial cache size while the later linear scaling reflects our limit on the minimum group size. We note that our tests are stressful in that groups 1-25 all have over 10 million users; *i.e.*, every domain has 25 groups with sources simultaneously multicasting the entire Internet.

In summary, caching should allow source border routers to handle forwarding in the line cards for at least several hundred thousand groups.

7. IMPLEMENTATION

We have built a prototype FRM router that runs under the Linux operating system using the eXtensible Open Router Platform (XORP) [46]. Figure 7 illustrates the overall structure of the FRM prototype. A Linux kernel module implements the FRM forwarding plane and a user-level component manages group membership state and propagates membership updates to neighboring ASes. The user-level module runs in the execution context of the XORP BGP daemon (`xorp_bgp`) and communicates with the kernel-side FRM module via the Linux netlink mechanism. At kernel level, the `FRMHdrCache` table caches forwarding state for groups that have sources in the router's local domain while the `BGPPeerTable` holds the encoded AS edges used to forward transit packets. The GRP_BFs are stored in the BGP RIB in XORP. Our prototype currently lacks support for interfacing FRM to intra-domain multicast routing protocols; instead, as an interim mechanism, we maintain a local table (`LocalGrpMembers`) that stores the IP addresses of local group members. A more scalable implementation might, for example, store the IP address of the group's local RP. We mod-

ify the designated router (DR) side of the IGMP implementation to insert/remove shim headers. Endhosts are thus unchanged and FRM routers only *update* shim headers. Our impementation adds 3500 lines of code to the Linux kernel and 1900 lines to the BGP daemon.

7.1 Packet Processing

The kernel delivers incoming multicast packets to the FRM module. If the source address indicates that the packet originated in the router's local domain, then we first check for forwarding state in the `FRMHdrCache` cache.

Source domain: cache miss. In the event of a cache miss, the kernel upcalls to `xorp_bgp` to request the multicast tree for the packet's destination group. `xorp_bgp` responds with a set of structures of the form AS_x : $SubTree_x$, where AS_x is the AS number of a direct child node and $SubTree_x$ is a list of edges in the subtree at AS_x. The kernel parses the daemon's response and constructs the FRM shim headers for every AS_x.

Our shim header consists of 32 control bits, followed by the TREE_BF. The first 4 control bits hold the number of bloom filter hash functions, followed by 4 bits for the length of the TREE_BF in multiples of 16 bytes. The next 16 bits carry a checksum computed over the shim header; the last 8 bits are currently left for future protocol extensions.

Once the headers are computed, a copy of the packet is made for each AS_x, its shim header updated appropriately, and then sent out. We use an auxiliary data structure (`BGPPeerTable`) in the kernel to map from the AS number of a BGP peer to its corresponding next-hop IP address. Finally, we add the destination group address and the set of shim headers for each AS_x into `FRMHdrCache`. The `FRMHdrCache` cache is indexed by group address and uses a basic LRU replacement scheme.

Source domain: cache hit. In the event of a cache hit, processing is simple – a copy of the packet is made for each AS_x entry associated with the destination group, the packet's shim header is updated with the appropriate shim header, and the packet sent to AS_x.

Transit domain processing. If the packet did not originate in the router's local domain, processing is straightforward: we decrement the IP TTL, update the IP checksum and finally traverse the `BGPPeerTable` checking for the presence of the edge denoted 'AS_{local} : AS_x' in the packet's FRM header. If present, we forward a copy of the packet to the next-hop address for AS_x. As the last step,a copy of the packet is sent to every local member listed in the `LocalGrpMembers` table.

We measure the forwarding latency for the above code paths. Our measurements were performed on a 1.8GHz IBM Thinkpad with 1GB RAM running FRM under Linux RedHat 9, kernel level 2.4.20-8. Table 4 lists the forwarding time for packets that hit in the `FRMHdrCache` cache under increasing fanout (*i.e.*, outgoing copies) for different payload sizes. Relative to unmodified Linux, FRM exhibits similar scaling behavior but is always slower in the absolute. Examination reveals this is primarily because our FRM implementation incurs one additional buffer copy for every packet sent – in standard multicast, an identical copy of the packet is sent to all outgoing next hops while generates a distinct copy of the packet (with appropriate shim header) for every neighbor and hence replicates the original buffer.

To measure the forwarding time for packets that suffer a cache miss, we populate the RIB with an Oct'04 snapshot of a BGP table with 117519 prefix entries and initialize a fraction of prefixes to indicate membership in the packet's destination group. Table 5 lists the forwading time for an increasing number of prefixes included in the tree. We see that, in the worst case where every prefix has a group member, it takes approximately 303.2 ms to forward the packet. Further investigation revealed this time is dominated by the cost of the BGP RIB scan. Although clearly expensive, we do not view the processing latencies of cache misses as cause for concern due to two reasons: First, these measured latencies are entirely dependent on the processor speed and other hardware characteristics of the router which is, in our case, a uniprocessor IBM Thinkpad. In reality, header construction can be parallelized and optimized on SMPs. Second, this latency is only incurred on the first packet sent to a group, and can be rendered even more infrequent by avoiding cache misses through pre-computation and an appropriate choice of the cache size.

Finally, Table 6 lists the forwarding latency for transit packets for different tree fanout values and different sizes of the table `BGPPeerTable`. We observe that transit forwarding is efficient and only marginally more expensive than a cache hit at the source router for the same tree fanout. As with source forwarding, the processing time scales linearly with the number of outgoing packet copies. As expected (given our software implementation) the results are linearly dependent on the domain's AS degree though TCAM would avoid this.

In summary, the design of FRM admits a straightforward implementation of the cache hit and transit forwarding code paths that achieve efficiency comparable to that of the native kernel forwarding. For cache misses, we believe a combination of hardware and software optimizations, along with a sufficient cache memory allotment can make the performance impact of misses negligible but an exploration and evaluation of performance optimizations merits further study, particularly in the context of realistic router forwarding engines.

Fanout	Linux mcast 1-byte pkts	FRM 1-byte	FRM 128-bytes	FRM 1024 bytes
1	0.4	0.7	0.8	1.2
128	25.4	64.8	76.2	89.5
256	50.7	132.5	154.2	177.5
512	101.2	262.7	308.6	351.4

Table 4: ***Forwarding time (in μsecs) at R_s when the group is in*** `FRMHdrCache`**.**

#entries in tree	117519	29296	7300	1831	471	0
proc. time	303.2	124.8	89.1	74.5	68.3	65.8

Table 5: ***Forwarding time (in milliseconds) at R_s when the group is not in*** `FRMHdrCache`***. Packets are 512 bytes.***

Fanout $\Rightarrow$ AS deg. $\Downarrow$	1	32	128	256	512	1024
1	7.6					
32	10.9	38.8				
128	17.0	43.8	127.1			
256	27.7	54.5	137.1	220.7		
512	45.6	73.5	159.4	248.8	402.2	
1024	81.4	113.4	204.5	308.0	465.2	748.7

Table 6: ***Forwarding time (in μsecs) at R_t for 512-byte packets.***

7.2 Advertising group membership changes

An endhost's IGMP reports are delivered to its designated router (DR). In our current implementation, we modify DRs to relay these reports directly to the source FRM router R_s which updates its `LocalGrpMembers` table. We define a new optional transitive path attribute `FRM_UPDATE` for communicating incremental group membership changes and `FRM_GRP_BF` for the initial transfer of GRP_BFs at the start of a peering session.

To avoid a full scan of `FRMHdrCache`, we use an auxiliary data structure that efficiently resolves a bit position into a set of pointers to cached groups associated with that bit.

In our evaluations, the processing cost of an update message for a single group activation event that modifies 6 bits in the membership Bloom filter and invalidates a single `FRMHdrCache` entry (with 1024 entires present in the cache) requires total processing time of 18.6 μsec. It takes 0.34 μsec to update the Bloom filter and 18.33 μsec to perform the invalidation.

Finally, to test FRM end-to-end, we set up a local testbed of 4 interconnected FRM routers, with 2 Windows desktops running unmodified VAT [47] that connect to our FRM network via our modified DRs. We observed packet delivery from the VAT source to receivers demonstrating that FRM can forward packets end-to-end using legacy endhost stacks and applications.

8. DISCUSSION:

Usage model. It is likely that a multicast service deployed today, would not adopt an open usage model. We speculate on possible usage models but stress that issues of address allocation, access control and charging merit much greater scrutiny than we can provide here.

ISPs might control use of multicast at two levels – per-user and per-group. The first determines whether a user is allowed to send and/or receive multicast traffic (independent of which groups). As with unicast connectivity, users sign up with their local ISP for multicast service and the local ISP handles access control and charging of users. ISPs might distinguish between service offerings that allow users to both send and receive traffic from those that only allow a user to receive multicast traffic. For senders, ISPs might choose to charge based in proportion to the group size or include limits on the (AS-level) group size in the service agreement. FRM assists ISPs in this regard as it allows the access provider to accurately compute and control the extent of the dissemination tree.

Access control at the group level controls which group addresses are routable. ISPs might each be allocated a portion of the multicast address space and, to create a group, a user must explicitly obtain an address from some ISP. The role of the allocating ISP is merely to legitimize the group and does not constrain membership of the group in any way. ISPs only route group addresses that can be proven to have been legitimately allocated by a recognizable ISP. For this, an allocating ISP signs the group address with its private key; group members may retrieve this signature via the same channel (*e.g.*, DNS) used to discover the group address and can present the signature to its local ISP when it joins and/or sends to a group. To verify signatures, ISPs use the signing ISP's public key which can be disseminated along with an ISP's BGP adverts. Allocation of a group address can be associated with a fee and a lease period allowing prices to be driven by demand.

The above serves to limit service to legitimate users and legitimate groups but does not attempt to regulate which users are allowed access to which groups. We conjecture that this may be a tractable level of control for ISPs to implement while leaving more fine-grained access control to be handled by applications as per their (different) needs. At the same time, the above access control schemes could accommodate some extensions for more fine-grained control; *e.g.*, a user's service contract could limit the groups it may join or the allocating ISP's signature could include a list of authorized sender IP addresses.

Finally, while the above assumes ISPs control address allocation, this is not strictly required as FRM imposes no structural restrictions on the allocation and use of group addresses.

Attacks on the FRM protocol. With the above, malicious attempts to trigger frequent GRP_BF would be limited to legit groups which should make it harder to cause domain-wide fluctuations in membership. Moreover, this is tantamount to a user attacking its local ISP which increases attacker exposure. The same is true for malicious users that send to many different (valid) groups so as to burden routers with the more expensive tree construction operations.

Intra-domain FRM. FRM may be applied unmodified within domains that run link-state protocols. For domains with distance-vector-based protocols, FRM requires modification to work in the absence of complete path information. For this, we could encode destination nodes, as opposed to tree edges, in the shim header. As mentioned in section 5 this would require that intermediate routers repartition the set of encoded leaves to avoid duplicate forwarding though the results of this could be cached.

Relative to running intra-domain link-state MOSPF, FRM's source-encoded forwarding reduces the state and computational load at intermediate routers but requires a shim header. Admittedly, these are modest advantages and hence replacing intra-domain MOSPF by FRM would more likely be motivated by a desire for uniformity in intra and inter-domain solutions. Relative to intra-domain PIM-SM, FRM avoids the need to configure RPs.

Other routing services. As FRM makes no use of hierarchical address allocation or aggregation, its implementation represents a fairly general abstraction – subscription to, and location of – flat identifiers and could thus be applied to more general routing services such as IP layer anycast, data-centric or name-based routing. The main difference is that multicast requires matching *all* subscriptions while the above require matching *any*. The only implication to our design is that false positives would be undesirable; a simple solution would be to instead, enumerate subscriptions or use compression that admits only false negatives.

9. CONCLUSION

FRM represents a different approach to implementing multicast. It is simpler in the wide area (no distributed tree construction), easier to configure (no need to place RPs), and allows providers to work within the familiar BGP framework to handle inter-provider issues. These features come at a cost of reduced efficiency and greater demands on border routers; a tradeoff that we believe is worth exploring given technology trends.

FRM tackles a purely technical barrier to deployment and other barriers do exist. However, given the growing adoption of Internet broadcasting, massively multiplayer games, and other networked applications we conjecture the time may be right to revisit IP multicast and re-evaluate its chances.

10. ACKNOWLEDGMENTS

We thank Katerina Argyraki, Kevin Fall, Zhi Li, Timothy Roscoe and the anonymous reviewers for their valuable input that helped improve this work. We would also like to thank Hitesh Ballani for helpful discussions on exploiting router resources.

11. REFERENCES

[1] Stephen Deering and David Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[2] Yang hua Chu, Sanjay Rao, and Hui Zhang. A Case for End System Multicast. In *Proceedings of SIGMETRICS 2000*, CA, June 2000.

[3] Christophe Diot, Brian Levine, Bryan Lyles, H. Kassem, and D. Balensiefen. Deployment issues for IP multicast service and architecture. *IEEE Network Magazine. Special Issue on Multicasting*, 2000.

[4] Hugh Holbrook and David Cheriton. Ip multicast channels: Express support for single-source multicast applications. In *Proceedings of SIGCOMM '99*, Cambridge, MA, September 1999.

[5] ISC Domain Survey, January 2005.

[6] Craig Labovitz, Abha Ahuja, Abhijit Abose, and Farnam Jahanian. An experimental study of delayed Internet routing convergence. 2000.

[7] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shikh, and Jacobus van der Merwe. Design and Implementation of a Routing Control Platform. In *Proc. of NSDI*, 2005.

[8] E. Castronova. Network Technology, Markets and the Growth of Synthetic Worlds. In *Second Workshop on Network and Systems Support for Games (NetGames)*. ACM, May 2003.

[9] MMOGCHART. http://www.mmogchart.com , http://terranova.blogs.com/terra_nova/2003/10/growth_rates_of.html.

[10] Blizzard Entertainment. WoW Surpasses 5 Million Customers Worldwide. 2005. http://www.blizzard.com/press/051219.shtml.

[11] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The Effect of Latency on User Performance in Warcraft III. In *Second Workshop on Network and Systems Support for Games (NetGames)*. ACM, May 2003.

[12] J. Pellegrino and C. Dovrolis. Bandwidth Requirement and State Consistency in Three Multiplayer Game Architectures. In *Second Workshop on Network and Systems Support for Games (NetGames)*. ACM, May 2003.

[13] Blizzard Entertainment. World of Warcraft. http://www.blizzard.com.

[14] Synthetic Statehood and the Right to Assemble. http://terranova.blogs.com/2005/02/the_right_to_as.html.

[15] Microsoft IPTV Edition.

[16] Ion Stoica, Dan Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet Indirection Infrastructure. In *Proceedings of SIGCOMM*, August 2002.

[17] Bryan Ford. Unmanaged Internet Protocol: Taming the edge network management crisis. In *HotNets*, November 2003.

[18] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. In *Proceedings of NGC*, London, UK, November 2001.

[19] Hui, Chaintreau, Scott, Gass, Crowcroft, and Diot. Pocket switched networks and the consequences of human mobility in conference environments. In *Workshop on Delay Tolerant Networking*, 2005.

[20] Kevin Fall. A Delay Tolerant Networking Architecture for Challenged Internets. In *Proceedings of SIGCOMM*, August 2003.

[21] D. Waitzman, C. Partridge, and S. Deering. *Distance Vector Multicast Routing Protocol*. ARPANET Working Group Requests for Comment, DDN Network Information Center, November 1988. RFC-1075.

[22] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT) an architecture for scalable inter-domain multicast routing. Technical report, San Francisco, CA, September 1993.

[23] Bill Fenner, Mark Handley, Hugh Holbrook, and Isidor Kouvelas. Protocol Independent Multicast – sparse mode (PIM-SM): Protocol specification, October 2003. Internet Draft.

[24] B. Fenner and D. Meyer. *Multicast Source Discovery Protocol (MSDP)*. ARPANET Working Group Requests for Comment, DDN Network Information Center, 2003. RFC-3618.

[25] K. Kumar, P. Radolavov, D. Thaler, D. Alaettinoglu, D. Estrin, and M. Handley. The MASC/BGMP architecture for inter-domain multicast routing. In *Proceedings of SIGCOMM '98*, Vancouver, BC CANADA, September 1998.

[26] Dina Katabi. The Use of IP Anycast for Building Efficient Multicast Trees. In *Proceedings of Global Internet*, 1999.

[27] T. Bates et al. *Multiprotocol Extensions for BGP-4*. ARPANET Working Group Requests for Comment, 2000. RFC-2858.

[28] Radoslavov et al. *The Multicast Address-Set Claim Protocol*. RFC-2909.

[29] David Thaler and Mark Handley. On the aggregatability of multicast forwarding state. In *Proceedings IEEE Infocom*, Israel, March 2000.

[30] Pavlin Radoslavov, Deborah Estrin, and Ramesh Govindan. Exploiting the bandwidth-memory tradeoff in multicast state aggregation. Technical Report TR99-697, University of Southern California, 1999.

[31] Briscoe and Tatham. *End-to-end aggregation of multicast protocols*, 1997. Internet Draft.

[32] W. Fenner. *Internet Group Management Protocol, Version 2*. Internet Engineering Task Force, Inter-Domain Multicast Routing Working Group, February 1996. Internet Draft.

[33] Pankaj Gupta. *Algorithms for routing lookups and packet classification*. PhD thesis, Stanford University, December 2000.

[34] M. Waldvogel ad G. Varghese, J. Turner, and B. Plattner. Scalable high speed IP routing lookups. In *Proceedings of SIGCOMM '97*, Cannes, France, September 1997. ACM.

[35] Cisco Systems. *Cisco 1200 Series 3GigE Line Card*. (linecard with 512MB buffer and 256MB route memory).

[36] Cisco Systems. *Cisco 1200 Series One-Port OC-192 Line Card*. (reports 512MB route memory).

[37] Katerina Argyraki and David R. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *Proc. of USENIX Annual Technical Conference*, 2005.

[38] S. Keshav and Rosen Sharma. Issues and Trends in Router Design. *IEEE Communications Magazine*, May 1998.

[39] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S.Shenker. Off by Default! In *Fourth Workshop on Hot Topics in Networks*, November 2005.

[40] Content Addressable Memory Cypress Semiconductor. http://www.cypress.com.

[41] S. Ratnasamy, A. Ermolinskiy, and S.Shenker. Revisiting IP Multicast. Intel Research Technical Report.

[42] V. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: Findings and implications. In *Proceedings of SIGCOMM*, Stockholm, Sweden, August 2000.

[43] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points. In *Proc. of IEEE Infocom*, 2002.

[44] Route Views Project Page.

[45] Cisco Systems. *Access list configuration in Cisco's Gigabit Ethernet Interface*. (reports GigE module supports up to 256K TCAM entries).

[46] Handley, Kohler, Ghosh, Hodson, and Radoslavov. Designing Extensible IP Router Software. In *Proceedings of NSDI*, 2005.

[47] Van Jacobson and Steven McCanne. *Visual Audio Tool*. Lawrence Berkeley Laboratory.

IPv6 Hitlists at Scale: Be Careful What You Wish For

Erik Rye
University of Maryland

Dave Levin
University of Maryland

ABSTRACT

Today's network measurements rely heavily on Internet-wide scanning, employing tools like ZMap that are capable of quickly iterating over the entire IPv4 address space. Unfortunately, IPv6's vast address space poses an existential threat for Internet-wide scans and traditional network measurement techniques. To address this reality, efforts are underway to develop "hitlists" of known-active IPv6 addresses to reduce the search space for would-be scanners. As a result, there is an inexorable push for constructing as large and complete a hitlist as possible.

This paper asks: what are the potential benefits and harms when IPv6 hitlists grow larger? To answer this question, we obtain the largest IPv6 active-address list to date: 7.9 billion addresses, 898 times larger than the current state-of-the-art hitlist. Although our list is not comprehensive, it is a significant step forward and provides a glimpse into the type of analyses possible with more complete hitlists.

We compare our dataset to prior IPv6 hitlists and show both benefits and dangers. The benefits include improved insight into client devices (prior datasets consist primarily of routers), outage detection, IPv6 roll-out, previously unknown aliased networks, and address assignment strategies. The dangers, unfortunately, are severe: we expose widespread instances of addresses that permit user tracking and device geolocation, and a dearth of firewalls in home networks. We discuss ethics and security guidelines to ensure a safe path towards more complete hitlists.

CCS CONCEPTS

• **Networks** → **Network measurement**; **Network privacy and anonymity**.

KEYWORDS

IPv6, hitlists, passive measurement

ACM Reference Format:
Erik Rye and Dave Levin. 2023. IPv6 Hitlists at Scale: Be Careful What You Wish For. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3603269.3604829

1 INTRODUCTION

ZMap [19] revolutionized Internet scanning in 2013, enabling scans of the entire IPv4 Internet in under an hour. Since then, Internet-wide scanning has become one of the most powerful and commonly used tools for measurement researchers and practitioners, leading to previously inaccessible findings in security [4, 12, 15, 18, 55], topology discovery [9], IoT measurement [30], outage detection [53], and more.

Unfortunately, the continued, accelerating deployment of IPv6 represents an existential crisis for Internet-wide scanning. In contrast to IPv4, brute-force scanning of every IPv6 address is impossible owing to IPv6's exponentially larger address space (2^{128} compared to IPv4's 2^{32}).

Without the prospect of iterating over all IPv6 addresses, some scanning tools instead rely on "hitlists" that identify addresses that are likely to be active and in-use, and probe only those. Others have introduced IPv6 target generation algorithms that emit potentially-active IPv6 addresses as probing candidates; these models must be trained on *some* hitlist and are biased to the types of addresses contained in their training data. Thus, the larger and more representative these hitlists are, the more complete the view of the IPv6 Internet available to measurement efforts [58].

As a result, the "holy grail" of Internet scanning is a complete list of all active IPv6 addresses. There are ongoing efforts approximate such a list. To date, the largest public list is the *IPv6 Hitlist* [24, 75]. The IPv6 Hitlist uses a variety of active measurement techniques—namely ZMap6 [70] and Yarrp [9]—and target generation algorithms to discover new responsive IPv6 addresses.[1] Their most recent efforts in 2022 nearly tripled IPv6 Hitlist's size to a total of 8.8M addresses [75].

In this paper, we ask: what are the potential benefits and harms when IPv6 hitlists grow larger? To answer this question, we obtain the largest list of active IPv6 addresses to date: 7.9 billion addresses, 898 times larger than the current IPv6 Hitlist snapshot and 370 times larger than the Hitlist over the same time interval. Our corpus contains nearly 15 times more active IPv6 addresses than *all* of the active IPv6 addresses discovered in the IPv6 Hitlist's four-year history though it was collected in about 15% of the time. In contrast with the active measurements employed by the IPv6 Hitlist, we passively collected the nearly 8 billion active IPv6 addresses in our corpus by running a set of 27 geographically-distributed NTP servers as part of the NTP Pool [3], which devices worldwide use to synchronize their clocks.

Although our hitlist is not comprehensive—for instance, we are likely missing most modern Android devices because they are not configured to use the NTP Pool by default—it is a significant leap forward from the current state of IPv6 hitlists. As such, it provides a glimpse into what the future holds as the community moves toward its goal of a more comprehensive hitlist.

Our glimpse into the future of IPv6 hitlists produces two broad results: that network measurement insights are indeed improved through a bigger list (especially one that is passively collected like

ACM SIGCOMM '23, September 10, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604829

[1] We use capitalization to differentiate between hitlists in general (lowercase) and the IPv6 Hitlist in particular (uppercase).

ours), and that large hitlists poses significant security and privacy risks.

Benefits: New insights from larger hitlists What is gained from having larger IPv6 hitlists? What do current hitlists lack, and does filling in those gaps facilitate a deeper understanding of the IPv6 Internet?

We find that a significant missing portion of prior IPv6 hitlists is *end-host addresses*. The IPv6 Hitlist, for example, relies primarily on ZMap6 and Yarrp, and as a result discovers many infrastructure nodes (especially routers and CPE), but has difficulty identifying end-hosts due to firewalling and frequently unpredictable and ephemeral client addresses [50]. Conversely, our dataset is comprised largely of end-hosts, though not exclusively (virtually all Internet devices must synchronize their clocks).

We demonstrate that when a hitlist has more end-host addresses, it enables investigations that are impossible with existing hitlists, such as studying client address entropy, IoT detection, and address assignment patterns. However, these new insights are not without cost.

Threats: New privacy leaks from larger hitlists Conventional wisdom dictates that IPv4 addresses do not constitute personally identifiable information (PII). This is largely because the link between a device and an IPv4 address is very weak: addresses are assigned randomly and are frequently reassigned, and multiple devices often map to the same address at any time. However, the same is not always true for IPv6 addresses. It is now well-known that IPv6 addresses can risk uniquely identifying a client device—e.g., if the device embeds its MAC address into the lower-order bits of its IPv6 addresses, then it could potentially be tracked as it moves across networks.

These privacy issues were not a major concern for previous IPv6 hitlists, as they focused primarily on infrastructure nodes, which typically have no active user and do not move much across networks. However, more complete hitlists will inevitably include more client devices.

We perform what we believe to be the first analysis of the privacy risks inherent to large, client-rich IPv6 hitlists. We observe nearly 15 million clients in multiple networks across four distinct types of address tracking, and apply recent precision IPv6 geolocation techniques to hundreds of thousands of addresses.

Collectively, our results show that there is significant promise in store for larger IPv6 hitlists and Internet scanning, but also potential harm in making larger hitlists public.

Contributions To summarize, we make the following contributions:

- We collect and report on the largest publicly-obtained IPv6 hitlist to date: over three orders of magnitude larger than prior public hitlists.
- We perform a thorough comparison with the two premiere hitlists in use today—the *IPv6 Hitlist* [75] and CAIDA's routed /48 dataset [14]—finding that our NTP-derived dataset is far larger and comprises more end-hosts, but that each dataset provides a complementary perspective on active IPv6 addresses.
- We show that larger, more client-centric IPv6 hitlists enable new discoveries that prior hitlists do not, particularly in address patterns and aliased network discovery.
- We also show that larger, more client-centric IPv6 hitlists enable new *attacks* on privacy in the form of tracking and geolocation.
- We discuss the ethical ramifications of our findings, and provide guidelines that we hope will help shape future efforts in obtaining and sharing more complete IPv6 hitlists.
- We make the active /48 prefixes we discovered publicly available at `https://v6-research.cs.umd.edu`

2 BACKGROUND AND RELATED WORK

2.1 Why creating IPv6 hitlists is hard

Applications ranging from census and adoption studies [17, 52], to vulnerability identification and remediation [41], to outage detection [39, 54, 59, 67] all rely on an understanding of what IP addresses are assigned to live hosts. In IPv4, identifying these hosts is tractable due to stateless scanners like ZMap [19] and Yarrp [9] that can probe the IPv4 Internet in minutes. However, several factors complicate live address discovery and hitlist creation in IPv6.

The first and most obvious complicating factor is the immensity of the IPv6 address space. This leads to large prefixes—at least /64 and frequently /56 or larger [61]—being delegated to even residential customers. Thus, the average Internet subscriber has 4 billion or more times the number of IP addresses in the entire IPv4 Internet available to her, all of which are publicly routable; contrast this with IPv4, in which she commonly has one public IP, with her home network hidden behind a NAT. With such large prefixes delegated to customers with only tens or hundreds of devices, IPv6 is considerably more sparse than IPv4. Further, some service providers delegate prefixes to their customers for only short periods (e.g., 24 hours) before recuperating them and issuing new ones [64].

The lower 64 bits of a 128-bit IPv6 address is called the Interface Identifier (IID), and its assignment adds further complexity to creating IPv6 hitlists. Some IPv6 addresses are manually assigned: these are often infrastructure devices that network administrators prefer to assign easily memorable IIDs for troubleshooting [62], like `::1` or `::2`. Occasionally, some network operators will embed the IPv4 address assigned to the same interface in the IID [10], but there is no requirement to do so and these embeddings are relatively uncommon. Often, hosts self-assign IIDs via one of several processes. Extended Unique Identifier - 64 (EUI-64) Stateless Address Autoconfiguration (SLAAC) [49] embeds the interface's Media Access Control (MAC) address in the IID, after inverting the Universal/Local bit of the MAC and inserting a `0xFF 0xFE` in-between the third and fourth bytes. Because these addresses embed a static, link-layer identifier in the IPv6 address that allows for device identification and tracking, as well as attacks tailored to device manufacturers, generating ephemeral random addresses has instead been encouraged since 2001 [50]. However, ephemeral addresses are problematic for servers, which should possess stable yet privacy-preserving addresses for long periods of time. In response, standards have been proposed for generating stable, random addresses [28]. Finally, DHCPv6 [46], the less-ubiquitous cousin of its IPv4 analog, also exists to assign addresses to hosts. However, given such large prefix allocations, it is up to the DHCPv6

implementation and operator configuration how client addresses are assigned.

Even responsive addresses in IPv6 may not indicate a live host. In IPv6, *aliasing*, in which a single device replies to probes to an entire network, is a relatively common practice. This necessitates a filtering step in hitlist creation, wherein responsive addresses from aliased networks are removed.

2.2 Related Work

There have been extensive efforts to discover new live IPv6 addresses through both active and passive measurements.

Active approaches Beverly developed the Yarrp high-speed, stateless traceroute tool to improve host and topology discovery in both IPv4 and IPv6 [9], and Beverly et al. used Yarrp to discover significant IPv6 Internet core topology [11]. Similarly, Gasser et al. developed ZMap6 [70] IPv6 extensions of the ZMap [19] high-speed scanner to enable fast IPv6 scanning without tracing to intermediate hops [25]. Gasser et al. used a combination of ZMap6, scamper traceroutes [38], and public data sources to develop an IPv6 hitlist and identify aliased networks [24]. They continue to publish a weekly hitlist of responsive addresses and known aliased and non-aliased networks [1]. Rye and Beverly used Yarrp and ZMap6 to focus discovery of topology at the network periphery (Customer Premises Equipment (CPE) devices in customer ISP networks) [65] and characterized high-frequency customer network changes [64]. Others have enumerated reverse DNS zones to discover active IPv6 addresses [13, 21, 69]. Foremski et al. [22] aggregated datasets comprising more than 3.5 billion IPv6 addresses from cloud providers and ISPs to develop new candidate addresses for active measurements. Numerous machine learning models [68] have been trained on responsive addresses in order to generate candidate addresses for active measurements, using Bayesian networks [47], reinforcement learning [32], generative adversarial networks [16], divisive hierarchical clustering [37], and ensemble learning [72].

As we will demonstrate, our passive approach is largely complementary to these active efforts, exposing portions of the in-use IPv6 address space that the active efforts alone do not reach.

Passive approaches Numerous passive IPv6 efforts also have studied IPv6 addressing. Gasser et al. [24] and Huz et al. [33] both crowd-source small numbers of IPv6 client addresses via Mechanical Turk [6] and Prolific Academic [2]. A major barrier to conducting large-scale passive IPv6 measurements, however, is access to proprietary datasets. Plonka and Berger [56] use IPv6 addresses gathered from a large CDN's webservers to determine how customer addresses change over time. Using data obtained from Facebook, Li and Freeman [36] examine how client IPv6 addresses change over time and consider the problem of handling abusive or malicious actors in IPv6. Saidi et al. [66] examine aggregated IPv6 client traffic, including NTP, provided by a large European ISP to track customer subnet allocations over time by tracking clients employing EUI-64 IPv6 addresses. Enayet and Heidemann use data from the DNS B root to detect outages in IPv4 and IPv6 [20], while Fukuda and Heidemann use DNS backscatter from the B root to detect IPv6 scanning [23].

Comparison of passive and active approaches Several works compare active and passive measurement approaches, albeit only in IPv4 and on a far smaller scale than our work. Bartlett et al. compared passive monitoring and active probing to discover services running on a university network [8]; Heidemann et al. similarly used passive and active approaches to discover IPv4 end hosts in 2008 [31]. Zander et al. used several passive sources of active IPv4 addresses, including Wikipedia edits and NetFlow records, to augment active measurements to estimate IPv4 address space utilization [73].

Unlike prior active efforts, we avoid sending millions of unsolicited probes in search of active addresses. Unlike many prior passive efforts, our technique does not require access to privileged datasets obtained by private organizations. Rather, we demonstrate that enormous amounts of IPv6 data can be obtained by contributing to an open service—the NTP Pool—which we review next.

2.3 NTP and the NTP Pool

The Network Time Protocol (NTP) is one of the Internet's oldest protocols, standardized in 1985 in RFC958 [44]. NTP synchronizes a device's clock with a remote time server, even in variable-delay networks. Keeping accurate time is of immense importance to a variety of applications ranging from TLS certificate validation [40, 74], to authentication [35, 40, 45], to DNS cache entries [40]. Most devices on the Internet today synchronize their clocks using NTP.

Where a device looks for its time is typically a function of its operating system. Windows clients and servers, for instance, synchronize their time with `time.windows.com` [43] by default when not joined to a domain. Likewise, Apple devices synchronize with `time.apple.com`. Android clients until Android 8 (Oreo) used the NTP Pool (`pool.ntp.org`, discussed next); later versions now use `time.android.com` [27]. NTP servers can additionally be specified via DHCP [5] and DHCPv6 [26] options.

The NTP Pool Project [3] provides NTP service through a worldwide set of geographically distributed NTP servers, many of which are contributed by volunteers. Any host with a publicly reachable IP address can serve in the NTP Pool. The Pool preferentially directs clients to servers geographically near them, using a combination of IP geolocation and DNS round robin. The NTP Pool Project further provides various "vendor zones" to equipment and software vendors to use as defaults on their devices; vendor zones for Android, Ubuntu, and CentOS exist, among many others (`android`, `ubuntu`, and `centos.pool.ntp.org`, respectively.)

3 METHODOLOGY

This section first highlights the measurement infrastructure we established to conduct our experiments. Then, it introduces the other IPv6 address datasets that we compare to our NTP-derived corpus. Finally, we discuss how we geolocate NTP client addresses and our methodology for probing back to active NTP clients that query our servers.

Vantage Points In order to measure the effectiveness of using NTP servers as large-scale, longitudinal, passive IPv6 measurement infrastructure, we operated 27 NTP servers from 25 January 2022 through 31 August 2022. We chose Virtual Private Servers (VPSs) from 20 countries across 6 continents to obtain geographic diversity.

Dataset	Dates	IPv6 Addresses		ASNs		/48s		Avg. Addrs
		Num.	Common	Num.	Common	Num.	Common	per /48
NTP Pool (This paper)	Jan–Aug '22	**7,914,066,999**	–	9,006	–	7,205,127	–	**1,098**
IPv6 Hitlist [1]	Feb–Aug '22	21,409,629	277,026	**18,184**	7,560	431,851	267,908	50
CAIDA Routed /48 [14]	Feb–Apr '22	11,613,494	3,117	13,770	6,957	11,111,563	102,864	1

Table 1: Comparison IPv6 datasets considered. Our NTP corpus is passively collected, while the comparison datasets used active techniques. "Common" denotes the intersection of the comparison data with our data.

Specifically, we ran 6 servers in the US, 2 in Japan, 2 in Germany, and 1 server in each of: Australia, Bahrain, Brazil, Bulgaria, Hong Kong, India, Indonesia, Mexico, Netherlands, Poland, Singapore, South Africa, South Korea, Spain, Sweden, Taiwan, and the United Kingdom.

Though we ran servers in 20 countries, the NTP Pool's load balancing allowed us to collect data from 175 countries[2] in total. The majority of the IPv6 addresses we discovered came from India (1.9B), China (1.6B), US (1.2B), Brazil (700M), and Indonesia (630M), collectively accounting for 76% of our entire dataset. The other 170 countries accounted for 24%.

We emphasize that each VPS was minimally provisioned and cost on average *less than $7 per month* to operate. We typically used one virtual core, 500MB–2GB of RAM, and a Linux OS available from the VPS vendor (Ubuntu, Amazon Linux, or CentOS).

Each of these VPSs was configured as a stratum-2 NTP server and joined to the NTP Pool. Because the NTP Pool directs clients to its NTP servers via a DNS round-robin that incorporates the client's coarse-grained IP geolocation, our globally-distributed NTP servers were visited by a wide range of clients around the world.

Comparative Datasets In order to compare our passive NTP results with contemporaneous, state-of-the-art IPv6 measurements, we acquired two external datasets. First, we compare against the IPv6 Hitlist [24], which provides a list of responsive addresses and networks that the operators detect as being aliased networks (responsive on all addresses) and those that are not. The Hitlist is updated on roughly a weekly basis, so in order to best compare our dataset to their data, we consider all Hitlist responsive IPv6 addresses published during our study's time frame. IPv6 Hitlist data concurrent with our study was published first on 16 February 2022 and runs through 29 August 2022. While our study collects only NTP requests, the IPv6 Hitlist is obtained through active measurements using ICMPv6, TCP ports 80 and 443 (HTTP and HTTPS), and UDP ports 53 (DNS), 161 (SNMP), and 443 (QUIC). Our IPv6 Hitlist comparison dataset consists of 21,409,629 unique IPv6 addresses.

Second, we leverage a large dataset of 1,083,188,032 Yarrp traces conducted by CAIDA from their Archipelago distributed measurement system [34] between 3 February and 6 April 2022. Their measurement methodology splits each prefix of length /32 or longer into /48s and probes the `::1` address of each /48. For prefixes of length less than /32, only a single `::1` address is probed, with no splitting into constituent /48s. These traces discovered 11,613,494 live addresses. We refer to this measurement as the "CAIDA routed /48" dataset throughout.

[2]We count ISO-3166-1 two-letter country codes and use the term "countries," although some are dependent territories.

Table 1 lists the relevant details of each dataset involved in our study. We will explore these numbers in more detail in §4, but even at a glance it is clear that the NTP data corpus comprises three orders of magnitude more addresses, with a greater density of addresses on average in each /48.

Geolocation We consider two different types of geolocation in our results. First, we used MaxMind's GeoLite2 City database [42] to geolocate the NTP client addresses we observed. While fine-grained IP-geolocation is often error-prone, particularly in IPv6, we consider only the country reported by MaxMind in aggregated results and do not use the more granular geolocation data it reports.

Second, for IPv6 addresses with embedded MAC addresses in the form of EUI-64 Interface Identifiers (IIDs), we attempt to link this embedded MAC address with a wireless MAC address from the same device obtained from a geolocation service. EUI-64 IIDs are constructed by first inverting the seventh least significant bit of the most significant byte of the interface's MAC address. Then, a static `0xFF 0xFE` is inserted between the third and fourth bytes of the MAC address to create a 64-bit identifier; this is then used as the lower 64 bits of the IP address in an EUI-64 IPv6 address. Recovering interface MAC addresses is a simple process — the `0xFF 0xFE` bytes of an EUI-64 IIDs are removed, followed by the seventh bit's inversion.

Both Google and Apple both offer geolocation APIs [7, 29], and other individual and community projects also collect wireless geolocation information [48, 51, 57, 71]. Our methodology for linking wired EUI-64-derived MAC addresses to wireless MAC addresses follows that of Rye and Beverly [63], wherein they form a linkage between the most commonly-arising offsets between pairs of wired and wireless identifiers from within the same vendor-assigned three-byte address prefix, called an Organizationally Unique Identifier (OUI). This is often, but not always, the closest match between wired and wireless MAC addresses within the same OUI. This methodology is also limited to devices that have wireless and wired MAC addresses from the same OUI.

Backscanning In order to compare and contrast active and passive methodologies for compiling IPv6 hitlists, we actively probed NTP clients that visited five of our 27 NTP servers over the course of a week during January 2023. During this week, we recorded the source addresses of NTP clients that queried the servers over ten minute intervals. When the interval concluded, we initiated traceroutes from the servers back to the clients using Yarrp and sent ICMPv6 Echo Requests to the clients with ZMap6. The probe targets were both the NTP client address that had queried the NTP server, as well as a random IPv6 address within the same /64 as the client. All probes used ICMPv6 in order to minimize potential

disruption to the probed addresses; no IP was probed more than once during a 10 minute interval.

Ethical Considerations During this study, we accumulated nearly 8 billion unique IPv6 addresses by adding 27 NTP servers to the NTP Pool. Users of the NTP Pool had no way of knowing their NTP request data could or would be used in our study. That said, our study follows the same general principles of prior IPv6 Hitlist generation [24] as well as other peer-offered infrastructure, such as studies that use BitTorrent to collect and study IP addresses [60]. Like those studies, we do not collect any PII that might be included in the application-layer data (NTP requests do not contain PII).

Novel to our findings, however, is that large sets of IPv6 addresses may in and of themselves contain enough information to track and geolocate users. These attacks on privacy are made possible through the lower-order bits (specifically, we make use of EUI-64 IIDs). Thus, to avoid spreading this potentially sensitive information, we will only be releasing our dataset at the /48 level. This is an ethical consideration that future IPv6 hitlists must contend with: what is an appropriate way to share hitlists so as to enable Internet scanning tools to use them?

We communicated with the project owners of the NTP Pool to inform them of our experiments and to ensure that we were abiding by both the terms of service and acting in a way that preserved the privacy of the NTP Pool users. They concurred that we were not violating any NTP Pool policy or community standard and requested that address data released be aggregated to protect user privacy.

We also submitted our study to our institution's IRB for review. Much like other institutions' IRBs [24], they did not consider IP addresses as constituting human data. However, after informing them of our privacy results (§5), they agreed that further future consideration would be appropriate. Our hope is that this paper can be a first step towards the networking community helping to guide appropriate methods for ethically sharing IPv6 hitlists as they continue to grow.

Finally, contrary to active measurements that introduce immense volumes of superfluous data for the sole purpose of eliciting responses from remote devices (e.g., `traceroute` and `ping`), our experiments actually *provided a beneficial service* to the devices we measured. All of our servers provided stratum-2 NTP service and are located in cloud providers with exceptionally high availability, providing a reliable source of accurate time for NTP clients. Therefore, we believe that the benefits of our work outweigh the potential harm or risk that it may present.

4 BENEFITS OF LARGER HITLISTS

We begin our analysis by evaluating whether larger IPv6 hitlists confer benefits: as the community races to obtain larger hitlists, is it worth it? To this end, we first show that our NTP-based dataset is not only larger, but nearly disjoint with and complementary to other state-of-the-art IPv6 measurements. Then, we evaluate various applications of this larger hitlist: measuring aliased networks and analyzing IPv6 addressing patterns.

4.1 How Do the Datasets Compare?

First, we compare our dataset to the IPv6 Hitlist and a large-scale active measurement conducted by CAIDA.

In terms of size Table 1 compares the aggregate number of live IPv6 addresses we observed during our study to the number of responsive addresses collected by the IPv6 Hitlist project and during a large-scale, active measurement campaign by CAIDA. Running 27 IPv6 NTP servers from January through August 2022, we observed 7.9 billion unique IPv6 source addresses—370 times more than the IPv6 Hitlist's collections over a comparable period of time. The CAIDA measurement discovered 11,613,494 addresses during its two-month run, 681 times fewer than the NTP corpus.

In terms of addresses discovered Despite the massive difference in size, our dataset did not subsume either of the active datasets; we only discovered 1.3% (277,026) of the addresses that IPv6 Hitlist found, and a mere 0.02% of the IPv6 addresses CAIDA's routed /48 dataset discovered. This shows that the datasets are indeed complementary, and suggests that the kinds of devices we are finding are in fact distinct. We confirm this hypothesis later in this section.

In terms of Autonomous Systems (ASes) While the number of raw addresses in the NTP corpus dwarfs the other dataset address counts by several orders of magnitude, this trend is reversed in the number of ASes we observe. Our NTP corpus contains 65.3% of the number of ASes observed in the CAIDA scan (9,006 vs 13,770) and 49.5% of the number discovered in the IPv6 Hitlist data (9,006 vs 18,184). This discrepancy is likely due to the nature of the two active datasets, which use `traceroute`-like tools to discover Internet infrastructure between their vantage point and their probe targets. Our data, coming from NTP clients, is concentrated in ASes where NTP Pool clients exist, typically in customer ISPs. This hypothesis is strengthened by examining the types of ASes the different corpora addresses originate from, as classified by ASdb [76]. While the top AS type is consistent between all three datasets ("Computer and Information Technology","Internet Service Provider (ISP)"), an additional 14% (1,146,709,677) of our NTP Pool corpus originates from "Phone Provider" ISP subtype. By contrast, only 2% of the IPv6 Hitlist addresses originate from "Phone Provider" ASes, indicating that the NTP Pool corpus consists of a higher percentage of mobile clients than do either of the two active datasets.

In terms of prefixes Though the NTP Pool corpus IPv6 addresses are concentrated in fewer ASes than either of the two active measurements, our NTP dataset exhibits the highest number of addresses discovered per /48 (Table 1). The NTP dataset discovers a mean of 1,098 addresses per /48, while the IPv6 Hitlist uncovers 50 and the CAIDA scanning only 1. This "address density" has at least two potential root causes. First, NTP Pool clients may more commonly be client devices that frequently change their (random) IID in order to prevent tracking, as is considered best practice for client devices [50]. This phenomenon would manifest as many different addresses originating from the same prefix—for instance, a /56 or /64 allocated to a residential deployment by a customer ISP. A second possibility is that our passive NTP methodology detects more customer deployments than either of the active probing methods. While we detect devices in customer deployments so long as they visit an NTP Pool server, these networks are often highly subnetted

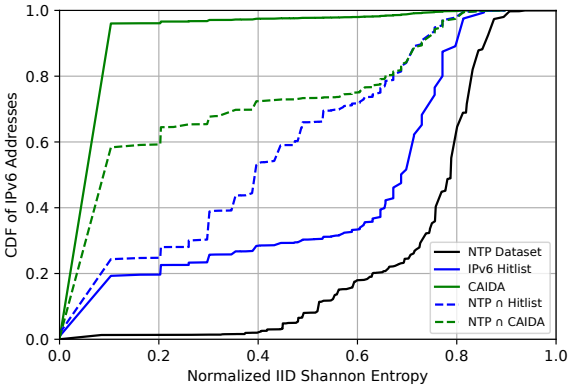


Figure 1: IID entropies of the IPv6 addresses from the NTP Pool, IPv6 Hitlist, and CAIDA routed /48 corpora, as well as the IID entropies of their intersections.

and would require significant active probing to discover the CPE devices, which may or may not respond. Neither the IPv6 Hitlist nor CAIDA scanning is specifically calibrated to do such fine-grained, residential deployment discovery. Note that these two root causes are not mutually exclusive; both may manifest in our data.

In terms of device type It is in general very difficult to perform device fingerprinting at scale. However, we can fortunately make use of features of the IPv6 addresses themselves to gain some insight into the kinds of devices that comprise the respective datasets.

We examined the IIDs, or lower 64 bits, of the addresses in each dataset. Because IIDs uniquely identify a host interface on a network, and because different types of hosts are subject to different concerns (e.g., preventing client tracking, static server addresses, or infrastructure addresses easily memorable by network administrators), the randomness (or lack thereof) of an IID may vary significantly between device types.

Figure 1 plots the CDF of all addresses found in each dataset versus IID entropy, using the normalized Shannon entropy as a metric. The NTP Pool corpus exhibits significantly higher entropy than the other two datasets, with a median normalized Shannon entropy of approximately 0.8. The IPv6 Hitlist has a somewhat lower median entropy of about 0.7, while almost the entirety of the CAIDA dataset has extremely low entropy.

These data points reinforce the hypothesis that *our NTP Pool dataset consists primarily of client devices*, which often use ephemeral, random addresses to defend against long-term tracking. The CAIDA dataset, by contrast, discovers mainly core Internet infrastructure, which is often manually addressed by operators with an incentive to create easily-memorable addresses. The IPv6 Hitlist occupies a middle ground of sorts, discovering both core Internet infrastructure, as well as some higher-entropy addresses assigned to CPE devices at the network periphery. In the next subsection, we further investigate address entropies within our dataset to illuminate differences in IPv6 addressing schemes between service providers.

Observed address durations The durations over which we observe distinct IPv6 addresses vary significantly, as shown in Figure 2. Figure 2(a) displays a CCDF of the lifetimes with which we observe all of the 7.9 billion addresses in our dataset. More than 60% of them are observed only once (a "lifetime" of 0 seconds in the plot). With purely passive measurements, we cannot determine whether this is because the devices had highly ephemeral addresses, or simply because they only contacted our (or anyone's) NTP servers only once. This demonstrates the importance of combining passive collection with active scans; so long as a device shows up even once at our servers (e.g., at boot-up), it can be used in subsequent backscanning.

At the opposite extreme, 95,780,865 (1.2%) IPv6 addresses are observed for a week or longer, 32,985,774 (0.4%) IPv6 addresses for a month or longer, and 2,218,998 (0.03%) IPv6 addresses for more than six months. Figure 2(b) is a CDF of the 670,737,407 unique IIDs we observed, binned by the entropy of the IID. This figure shows that while ~10% more of the low normalized entropy (< 0.25) IIDs appear only once in our corpus than medium or high entropy IIDs, low entropy IIDs are more likely to persist for long periods of time. In fact, 10% of all low entropy IIDs are observed for a week or more, as compared to 5% or less of the medium and high entropy IIDs.

Summary Collectively, these results show that our hitlist of IPv6 addresses has little overlap with prior datasets, and in particular adds more end-host devices than any prior efforts. This is a natural progression in IPv6 hitlists, but one that would have been much more difficult with purely active measurement-based approaches.

4.2 Backscanning and Aliased Networks

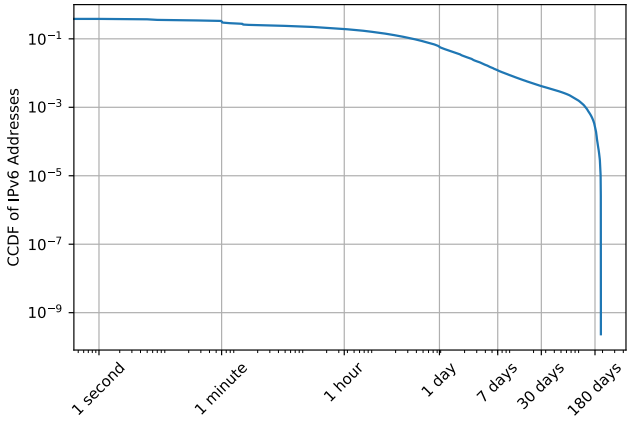
The IPv6 addresses we discovered are only useful for Internet scanning insomuch as they are responsive to outside scans. Moreover, if the addresses we learned are merely aliases of one another, then the volume of them would not be useful. To test both of these, we initiated Yarrp traces and ZMap6 probes from five NTP servers back to the clients that contacted them, as well as to a random address in the same /64 (see §3).

Responsiveness to backscanning As described in §3, we initiated a limited number of ICMPv6 scans back to NTP clients over the course of a week in January 2023. About two-thirds of the 71,341,581 NTP clients that were probed from the NTP servers responded to Yarrp or ZMap6 probes. This shows that the addresses we obtained can be used as scan targets.

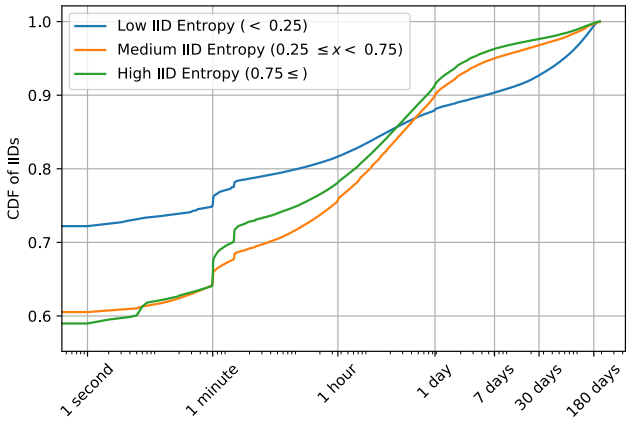
In contrast to the NTP clients we probed, the random targets we probed in the same network as the NTP clients responded only 3.5% of the time. Because it is exceedingly unlikely that we guessed a live random-IID IPv6 address, these responses almost certainly originate from aliased networks.

Figure 3 displays the IID entropy of the responsive addresses broken down by whether the address was responsive to backscanning ("NTP hit") or not ("NTP miss") or if the address was randomly chosen yet responsive ("Random"). The responsive NTP client addresses exhibit the lowest median normalized IID Shannon entropy, although they still exhibit higher entropy than addresses in our comparison datasets.

The higher the IID entropy, the less likely the end-host was to respond. Nearly 70% of the unresponsive NTP clients had normalized entropy greater than 0.75, compared to only ~50% of the responsive



(a) CCDF of distinct address lifetimes over all IPv6 addresses observed during our study.



(b) CDF of IID lifetimes by IID entropy category.

Figure 2: Address and IID lifetimes vary significantly. While most addresses and IIDs are observed only once, many persist for days or months.

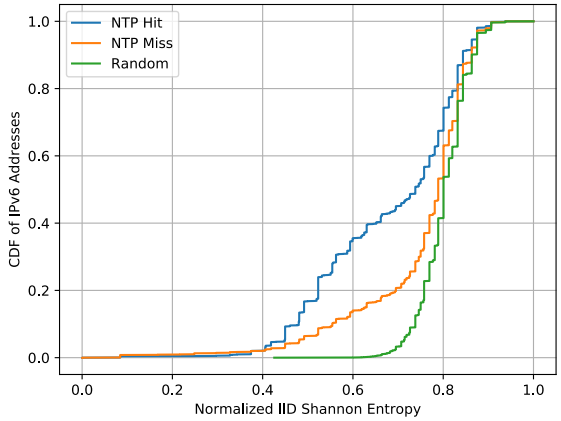


Figure 3: CDF of the IID entropy of NTP clients probed back with Yarrp and ZMap6. The clients that we passively detect from running an NTP client are responsive to back scanning, particularly those with slightly lower entropy in their IID.

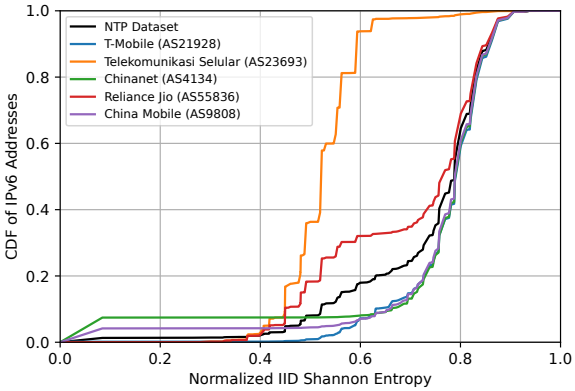
clients. We speculate that this can be attributed to infrastructure devices, which typically have stable, low-entropy IIDs, and are more likely to be responsive than client devices, such as mobile phones and personal computers. Client devices are more likely to reside behind a router or CPE device, which are often configured to block unsolicited inbound traffic, such as our ICMPv6 Yarrp and ZMap6 probes. Also, because IPv6 client addresses are often ephemeral, it is conceivable that some clients change addresses after querying our NTP servers and are no longer assigned that address when we probe it after the ten minute interval expires. Mobility may also play a factor, with devices switching to a new network in the period between sending an NTP request and when our probing began.

Discovering aliased networks As part of our backscanning, we received ICMPv6 responses to 4,476,089 unique, random IPv6 addresses we probed. These responses originated from 3,740,619 unique /64s. Because these addresses were chosen randomly from within active /64s, it is much more likely that the network in question is aliased rather than we randomly chose the address of a live host in an unaliased network. We compared our inferred aliased networks to those within the IPv6 Hitlist, which maintains a list of aliased networks. Of the 4.5 million aliased addresses we probed, the IPv6 Hitlist also categorized 4,425,001 (98%) as aliased. However, we discover an additional 46,512 aliased addresses in prefixes the IPv6 Hitlist does not recognize as aliased. This suggests that backscanning NTP clients is a potential avenue for discovering additional aliased IPv6 networks.

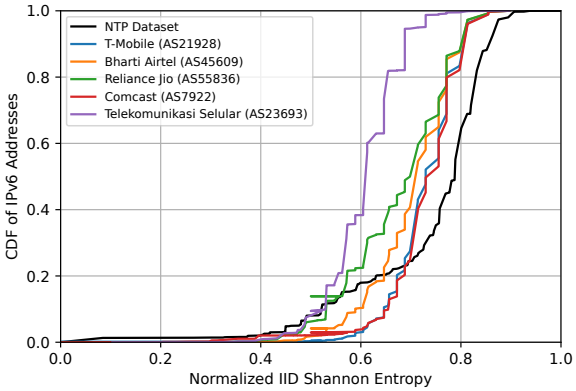
Finally, we examine the NTP clients in networks that we later determined were aliased via our backscanning technique. We find 3,841,751 NTP client addresses are part of aliased /64s as discovered by backscanning. These NTP client addresses originate from 36 different ASes.

We note that because the prefixes these NTP clients originate in are aliased, an active measurement campaign would be unable to distinguish these live hosts from aliased responses. That is, if it even attempted to scan the prefix in the first place—filtering known aliased networks is a best practice first step when conducting active measurements. Indeed, we searched for these addresses in a contemporaneous IPv6 Hitlist and found *only 23* addresses in our backscanning-detected /64s, while our NTP corpus contains 3,841,751.

Summary These results show that the majority of the addresses learned from our NTP-based dataset are responsive to scanning, despite the fact that most of them are clients and thus likely behind CPE devices. While this is a boon for Internet scanning, the fact that many of the devices are clients means that it is also a potential security issue; most would likely benefit from being behind firewalls. We also find that many of the addresses in our dataset are unlikely



(a) CDF of IPv6 address IID entropies observed in the top five ASes observed between Jan-Aug 2022.



(b) CDF of IPv6 address IID entropies for the top five ASes observed on 1 July 2022.

Figure 4: A comparison of the normalized Shannon entropies of collected IPv6 addresses over two time periods.

to ever be discovered with today's active measurement techniques: guessing active random IPv6 addresses or differentiating active addresses in aliased networks is impractical. Collectively, these results motivate complementing active measurements with passive data collection to obtain more complete IPv6 hitlists.

4.3 IPv6 Addressing Patterns

Our collection of IPv6 addresses permits insight into address allocation patterns, both on a macro scale, which has been previously studied [22, 24], and also at an AS level. The magnitude of addresses we obtain and the breadth of the networks from which they originate allows us to observe interesting phenomena specific to individual service providers. These phenomena are not visible from active measurements, and may help inform active measurement studies by illuminating the types of addresses present in various ASes.

Figure 4 displays two plots; each is a CDF over the total number of IPv6 addresses observed during a time interval plotted against the normalized Shannon entropy of address IID. While an imperfect proxy for randomness (e.g., the IID `0123:4567:89ab:cdef` has a normalized Shannon entropy of 1.0, but such a clear pattern might conceivably appear as the result of a network operator manually assigning it), IID entropy allows us to make generalizations about the types of addresses found within specific ASes.

Variability in entropy across ASes Figure 4(a) displays CDFs of the normalized IID Shannon entropies of the top five most commonly-observed ASes in our entire dataset, collected between January and August 2022. Three of the top five ASes (T-Mobile, ChinaNet, and China Mobile) exhibit entropy behavior that closely tracks with the NTP aggregate curve shown in Figure 1. The remaining two entropy curves, representing Reliance Jio and Telekomunikasi Selular (an Indian and Indonesian mobile provider, respectively), exhibit much lower median entropies. Further, while about 60% of the Reliance Jio addresses have high entropy (>0.75), approximately one-third exhibit a lower entropy below 0.6. This indicates that there are perhaps multiple classes of IPv6 addresses reaching our NTP servers from this network. Closer inspection reveals that at least two addressing patterns exist for hosts on Reliance Jio's network: one that randomizes all eight bytes of the IID, and another that uses the only lower four IID bytes with the remaining four set to 0.

Addressing strategies This result inspired us to investigate different patterns of addresses within ASes. To permit comparison to IPv6 Hitlist data, which is released in snapshots at intervals, we limit this analysis to a single day: 1 July 2022. Limiting this analysis to a single day also minimizes the influence that numerous random, ephemeral addresses from the same host might have over longer observation windows. Figure 4(b) shows the entropy of the top five ASes from that day in our dataset.

We compare the 46,195,900 NTP addresses we collect on 1 July 2022 to the IPv6 Hitlist's 2,970,366 IPs released on 1 July for the week prior across seven categories: (1) All zero IIDs ("Zeroes"); (2) IIDs with only the least significant byte set ("Low Byte"); (3) two least significant bytes ("Low 2 Bytes"); (4) IPv4 mapped addresses; and (5) high-entropy (>0.75); (6) medium-entropy (between 0.25 and 0.75); and (7) low-entropy (<0.25) IIDs. For IPv4 mapped addresses, which embed an IPv4 address in the IPv6 IID, we check whether any of three different embedded address encodings produce an IPv4 address in the same AS as the IPv6 address they are embedded in. We accept IPv4 embedded addresses only when i) there are at least 100 instances of them in the AS, and ii) more than 10% of the AS's total addresses are IPv4 embedded. These steps reduce false positives from random IIDs that coincidentally produce embedded IPv4 addresses in the same AS as the IPv6 address.

Figure 5 compares the frequency of these seven categories between our NTP-derived dataset and the IPv6 Hitlist. We find that the distributions of address types vary significantly between the two datasets. For the day considered, the NTP dataset is two-thirds high-entropy, with an additional 21% medium entropy. The IPv6 Hitlist, on the other hand, is approximately 20% medium and high entropy over the same time period. The fraction of IPv6 Hitlist Low

Manufacturer	Count	Manufacturer	Count
Unlisted	126,789,603	Sunnovo International Limited	1,193,746
Amazon Technologies Inc.	19,090,527	Hui Zhou Gaoshengda Technology Co.,LTD	1,067,459
Samsung Electronics Co.,Ltd	2,683,846	Huawei Technologies	876,083
Sonos, Inc.	1,633,209	Shenzhen Chuangwei-RGB Electronics	861,122
vivo Mobile Communication Co., Ltd.	1,330,987	Skyworth Digital Technology (Shenzhen) Co.,Ltd	723,316

Table 2: Number of MAC addresses extracted from EUI-64 IPv6 NTP clients by manufacturer (N = 171,611,786)

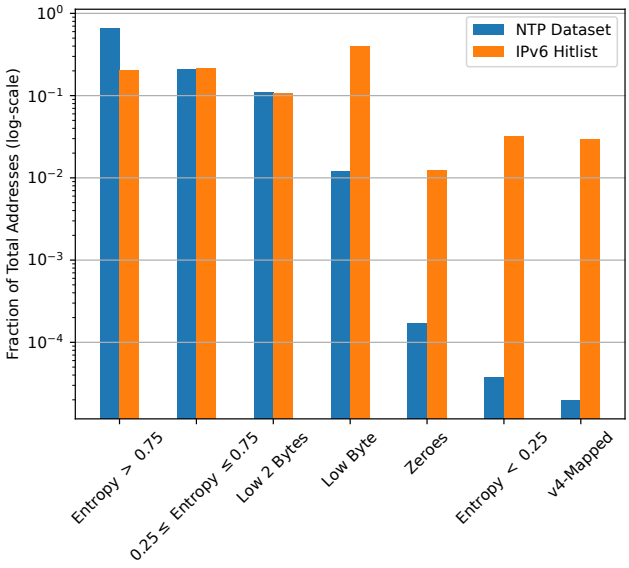


Figure 5: Fraction of the NTP corpus and IPv6 Hitlist that fall into each of seven categories for 1 July 2022. Note: y-axis is log-scale.

Byte addresses, however, is nearly 33 times that of the NTP corpus, and it contains 3% IPv4 mapped addresses compared to the NTP corpus' 0.00002%.

Taking only the IPv6 Hitlist into consideration, it would appear that the preponderance of IPv6 addresses' IIDs have only the least significant bytes set ("Low Byte"). But in our NTP dataset, the majority of active IIDs are in fact high-entropy. We believe that the Low Byte are more likely to be routing infrastructure; it is far easier for a network operator to manually set, read (e.g., in logs), and remember an IID with few bytes set (e.g., `::100`) than random ones. Indeed, because the IPv6 Hitlist relies heavily on `traceroute`-like techniques, we suspect it comprises many such routers. Nevertheless, the reality of the IPv6 Internet is that the majority of addresses are randomly set by clients: precisely the kinds of addresses that our prior results show are impractical for active measurements to obtain at scale.

Summary This example application of our NTP-derived dataset shows that *larger, more client-rich hitlists stand to improve future network measurement studies.* The push for larger hitlists is justified, at least from a measurement perspective. In the next section, we turn to whether larger hitlists come at increased security cost.

5 PRIVACY ISSUES OF LARGER HITLISTS

In this section, we perform what is, to our knowledge, the first empirical analysis of the privacy leakages in large, public hitlist datasets. We study two sources of privacy leakage: tracking via EUI-64, and geolocation by matching MAC addresses to geolocated BSSIDs (Basic Service Set Identifier, the MAC address of a WiFi access point), as in Rye and Beverly [63]. Our analysis here shows the potential harms inherent in larger hitlists.

5.1 Prevalence of EUI-64 IIDs

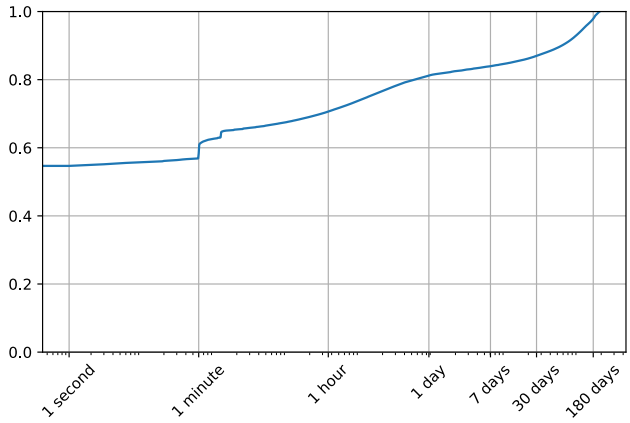
Both of the techniques we use to track and geolocate users make use of EUI-64 IIDs. EUI-64 IPv6 addresses have long been considered privacy vulnerability and have recently been studied extensively in CPE devices [63, 64] and in traffic from an ISP [66]. As our corpus consists primarily of client devices (§4) from a global network of NTP server vantage points, we were optimistic that EUI-64 IPv6 address prevalence would be low. Unfortunately, this was not the case.

Our dataset contains 238,281,703 EUI-64 IPv6 addresses: 3% of our corpus and more than the total number of *all* IPv6 addresses reported in our comparison datasets (see Table 1). Moreover, we can be certain that these are not randomly-generated addresses that *appear* to be EUI-64 due to `0xFF 0xFE` in the fourth and fifth bytes of the IID. The probability that a randomly-generated IID matches those bytes is 2^{-16}. Therefore, we would expect $\frac{7,914,066,999}{65,536}$ randomly-generated apparent EUI-64 IPv6 addresses, which is less than 121,000.

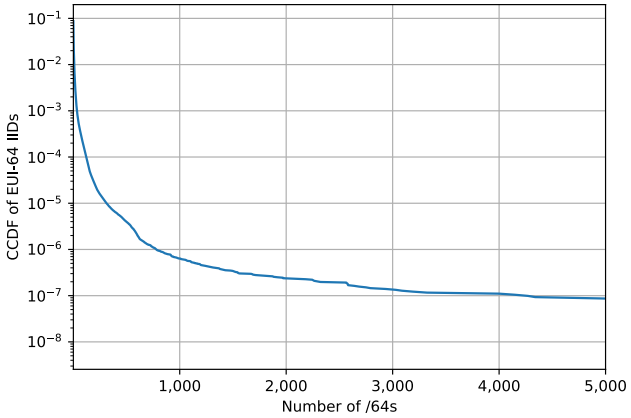
Among the EUI-64 IPv6 addresses present in our corpus, we find 171,611,786 unique embedded MAC addresses. MAC addresses can appear in multiple different IPv6 addresses for several reasons. First, when a device that uses EUI-64 addresses changes networks (e.g., due to mobility or a prefix rotation), its address will change but its EUI-64 will remain the same. Second, some device manufacturers have reused MAC address space, or assign common patterns (e.g., `00:00:00:00:00:00`), which then appear in EUI-64 IPv6 addresses for multiple devices.

To better understand the types of devices using EUI-64 addresses in our corpus, we first resolve the OUI from the MAC address extracted from the EUI-64 IID to the manufacturer listed in the IEEE's OUI database. We do this by removing the `0xFF 0xFE` from bytes four and five of the IID, and then inverting the Universal/Local bit of the resulting MAC address (the second-least significant bit of the first byte) if it is set. Table 2 contains the 10 most frequently observed manufacturers.

Surprisingly, the most common OUI we observe (126,789,603 MAC addresses, or 73.9% of all observed MACs in our dataset) is "Unlisted"—that is, they could not be resolved to a manufacturer at



(a) Lifetime of observed EUI-64 IIDs during the seven months of our study (x-axis logscale).



(b) CCDF of EUI-64 IIDs depicting the number of /64s each EUI-64 IID appears in.

Figure 6: EUI-64 IIDs permit long-term tracking of devices as they transition between network prefixes.

all. These are not merely random addresses; recall that we would expect fewer than 121,000 random IIDs to look like EUI-64. Moreover, manually inspecting the "Unlisted" addresses, we see significant MAC address counts in OUIs that are not in the IEEE OUI database. For instance, the most common "Unlisted" OUI is `F0:02:20`, with 52,218 distinct MACs embedded in EUI-64 addresses. The frequency with which this OUI appears in EUI-64 addresses makes it unlikely that these MAC addresses appear through a random process. On the other hand, 42,901 OUIs we classify as "Unlisted" appear in only one MAC address embedded in an EUI-64 address; we believe these are the randomly generated IIDs that appear to be EUI-64.

Included among the other nine most common manufacturers are makers of popular mobile, smart home, and IoT devices.

5.2 Tracking EUI-64 IIDs

Here, we evaluate the extent to which the EUI-64 IIDs in our dataset could be potentially used for tracking users.

Figure 6(a) depicts a CDF of the lifetime of all EUI-64 IIDs over the seven months of our study. While this generally tracks with all IIDs (see Figure 2(b)) EUI-64 IIDs are less likely to be observed only once (~55% compared to 60–70%) and exhibit the same long, fat tail that low-entropy IIDs do. This is due to the fact that when devices using EUI-64 change networks, they continue to use the same EUI-64 IID.

The lengthy observation window we observe for many EUI-64 IIDs demonstrates that *a passive adversary can leverage a large, longitudinal hitlist to track an unsuspecting user's device.* Because many service providers frequently "rotate" prefixes delegated to customers, often on timescales on the order of days or weeks, the ability to track a device by EUI-64 passively offers a major advantage over active techniques.

Figure 6(b) displays a CCDF of the number of /64s each EUI-64 in our corpus appears in. While most EUI-64 IIDs appear in only one /64 prefix, many appear in dozens, hundreds, or even thousands of /64s during the seven months of our study.

To determine which of the EUI-64 IIDs could be trackable users, we apply the following heuristics-based approach. We compute, for each EUI-64 IID: (1) The number of ASes it appears in; if more than 1, then we call it "high," otherwise "low." (2) The number of countries it appears in; if more than 1, then we call it "high," otherwise "low." (3) The number of transitions between different /64s it makes; if more than 10, then we call it "high," otherwise "low." If a device never changes its /64, then we consider it not trackable; of the 171,611,786 EUI-64 MAC addresses we observed, 14,943,429 (8.7%) of them appear in at least two /64s.

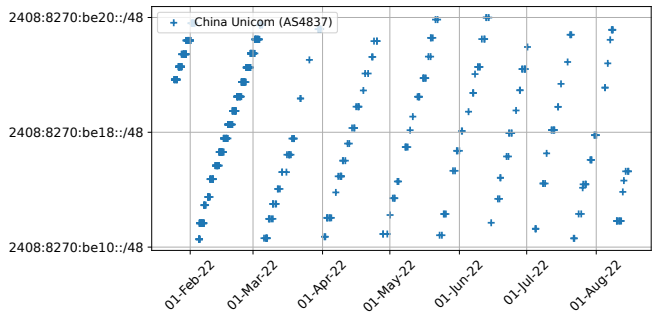
Using this heuristic, we classify EUI-64 IIDs into five categories as to the likely explanation for their re-occurrence:

Mostly static hosts The most common classification (12,853,055 of 14,943,429, or 86%) is EUI-64 IID that are labelled "low" across all three categories. These IIDs stay within the same AS and country throughout our observation period of them, and if they change /64s, they do so relatively rarely.

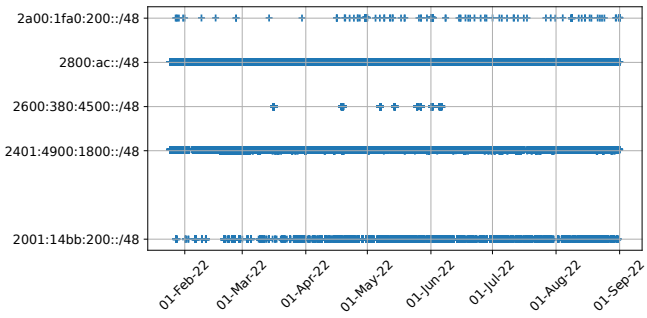
Likely prefix reassignment The second-most common classification (1,215,400 or 8%) appear in only one AS and one country, but /64 transitions is "high." One potential cause of this is service providers periodically reassigning new delegated prefixes to their customers Because this behavior is often based on provider policy, we observe it occurring more frequently in some providers than others. Figure 7(a) displays an exemplar of this behavior.

Likely MAC reuse In some instances (2,320 or 0.01%), we detect a single EUI-64 IID in a large number of ASes and countries, accompanied by a high number of transitions between /64. In these cases, we believe that are observing instances of MAC address reuse by a manufacturer, and that we are detecting several devices within different networks simultaneously. Figure 7(b) depicts a MAC address from EUI-64 IIDs that appear in a "high" number of countries and ASes.

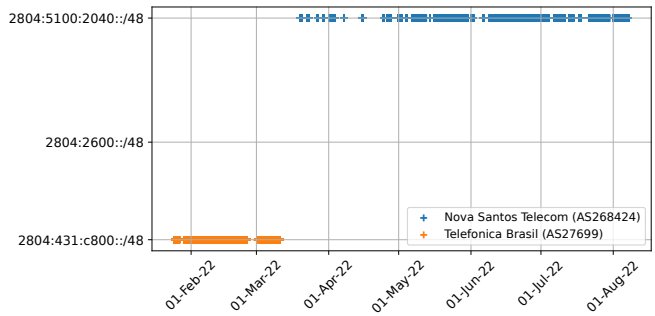
Changing providers We observe 5% of devices in multiple ASes within the same country that transition a "low" number of times between /64s. This behavior could arise from a static IoT or CPE



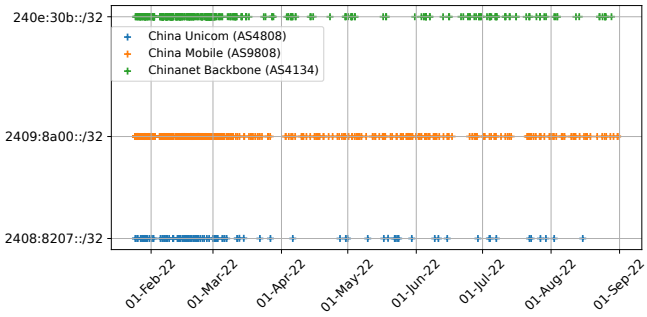
(a) A MAC from an unregistered OUI (`A8:AA:20`) is frequently renumbered within the same AS.



(b) A single MAC address appears in EUI-64 IPv6 addresses in 70 different ASes.



(c) A device changing service between two Brazilian providers.



(d) A Huawei MAC address frequently moves between multiple Chinese ASes.

Figure 7: A variety of EUI-64 IID tracking situations arise through the use of EUI-64 IPv6 addresses.

device changing service providers. Figure 7(c) displays an example of an EUI-64 IID seen transitioning between one AS to another. For the first month and a half of our study, the device appeared only in Telefonica Brasil's network. Mid-March and later, however, the device appeared only in Nova Santos Telecom.

Likely user movement Finally, 66,187 (0.44%) EUI-64 IIDs are observed in a high number of ASes within the same country that are also classified as "high" in /64 transitions. One possible cause of this behavior is a mobile device using EUI-64 addressing; as it transitions from a home WiFi network to a cellular network, it appears in multiple ASes and frequently changes /64s. Figure 7(d) depicts a Huawei MAC address that moves between three Chinese networks frequently over time. We believe that these are indicative of addresses that permit user tracking over time, and thus pose a risk to users' PII. While low in percentage, the raw number of potential user-tracking events this permits is large and concerning.

5.3 Geolocation

Recently, Rye and Beverly [63] described a technique for geolocating CPE routers by linking MAC addresses seen in EUI-64 IIDs with BSSIDs in publicly available wardriving datasets. Here, we apply their technique to all of our EUI-64 addresses. Whereas their initial analysis relied on active scanning and thus comprised mostly CPE routers, our dataset allows us to also consider IoT and mobile devices within customers' LANs (see Table 2).

We query geolocation databases (e.g. WiGLE [71] and Apple and Google's WiFi Location APIs [7, 29]) for WiFi BSSIDs in the same OUIs as the 171,611,786 MAC addresses we derive from EUI-64 IIDs. This produces 2,692,307 distinct WiFi BSSIDs with associated geolocation data. Next, we use our EUI-64 MAC addresses and wireless geolocation data to infer the offsets between wired and wireless MAC addresses in the same OUI. For each MAC address embedded in an EUI-64 IID, we compare it to each wireless BSSID in the same OUI from our dataset, recording the offsets between each pair of two identifiers. We then tally the most common positive and negative offsets from the wired MAC to a geolocated BSSID, and select the offset with the largest number of MAC-to-BSSID matches as the "correct" offset. In this manner, we generate wired-to-wireless offsets for 117 OUIs with at least 500 wired MAC-to-BSSID pairs. Finally, we used these offsets to determine how many of the EUI-64 MAC addresses we could match with WiFi BSSIDs and therefore geolocate.

All together, our methodology links 225,354 unique MAC addresses from our collected NTP dataset with geolocated WiFi BSSIDs. Although we do not have ground truth for the geolocations of these EUI-64 IPv6 addresses, we note that prior work validated the efficacy of this technique with a large US residential ISP [63].

Our geolocations resolve to 140 different countries. However, a large majority (174,155 or 75%) of the geolocated EUI-64 IPv6 addresses are from Germany. Mexico (7%), India (4%), France (3%),

and Luxembourg (2%) round out the top five countries of the geolocated EUI-64 IPv6 addresses. The over-representation of Germany and neighboring countries is due to the preponderance of AVM GmbH, the maker of the popular Fritz!Box router, MAC addresses in our geolocated EUI-64 IPv6 address set. AVM MAC addresses are responsible for 180,727 (80%) of the geolocated EUI-64 IPv6 addresses. Prior communication with AVM product security personnel confirmed this geolocation vector exists, and Fritz!OS version 7.50 eliminated EUI-64 WAN addresses support in December 2022.

Due to the prevalence of client addresses in our passive NTP corpus, using Rye and Beverly's CPE router geolocation technique permits fewer EUI-64 IPv6 geolocations than did their original study [63]. That work specifically targeted CPE with active measurements. However, we demonstrate that our NTP dataset contains some number of CPE routers that visit the NTP Pool for time that are susceptible to this privacy attack. Further, it is entirely passive. The only defense from this form of geolocation and tracking is to sever the linkage between the MAC addresses that appear in an EUI-64 IPv6 address and the BSSIDs that the WiFi access points use. Due to the potential for device tracking detailed earlier in this section, we recommend the use of random IPv6 addresses.

6 CONCLUSIONS

In this work, we accumulated the largest hitlist of IPv6 addresses solely through publicly-available means, without the aid of a CDN or ISP. We collected 7.9 billion unique addresses from a distributed set of 27 NTP servers located in cloud providers around the world. In addition to containing orders of magnitude more live addresses than existing hitlists, our dataset differs from current lists in that it contains *different types of addresses*. The addresses we obtain are highly entropic and ephemeral. They often come from client devices, as the OUIs of the embedded MAC addresses in EUI-64 IPv6 addresses show. And, crucially, these addresses are almost entirely absent in state-of-the-art hitlists today, which biases these hitlists toward addresses that can easily be discovered with active measurements or the DNS, like infrastructure devices and servers.

The ability to capture massive numbers of active client devices raises ethical questions not previously raised by active measurements. Since many of these addresses are highly random and ephemeral, they are likely uniquely associated with a single device or individual at a specific moment in time. Because of this uniqueness, we believe that these addresses deserve a special level of care in handling. As such, we will release our dataset truncated to the /48 level.

Finally, we repeat a plea for manufacturers to discontinue the use of EUI-64 IPv6 addresses. Our results show that EUI-64 addressing is not uncommon among NTP clients, and is implemented by popular manufacturers of IoT and smart home products. The use of these addresses permits, in some cases, tracking of devices across networks, as well as fine-grained geolocation through correlation with wireless identifiers.

ACKNOWLEDGMENTS

We thank Ask Bjørn Hansen and Steven Sommars from the NTP Pool project for their support and feedback on data sharing considerations. We also thank Rob Beverly and Justin Rohrer for early feedback and infrastructure support, as well as the anonymous reviewers for their helpful comments. This work was supported in part by NSF grants CNS-1901325 and CNS-1943240.

REFERENCES

[1] 2023. IPv6 Hitlist Service. `https://ipv6hitlist.github.io/`.
[2] 2023. Prolific Academic. `https://www.prolific.co/`.
[3] 2023. The NTP Pool Project. `https://www.ntppool.org/en/`.
[4] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. 2015. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *ACM Conference on Computer and Communications Security (CCS)*.
[5] S Alexander and R Droms. 1997. DHCP Options and BOOTP Vendor Extensions. RFC 2132. `http://www.ietf.org/rfc/rfc2132.txt`
[6] Amazon. 2023. Mechanical Turk (MTurk). `https://www.mturk.com/`.
[7] Apple. 2023. Location Services and Privacy. `https://support.apple.com/en-us/HT207056`.
[8] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. 2007. Understanding Passive and Active Service Discovery. In *ACM Internet Measurement Conference (IMC)* (San Diego, California, USA) *(IMC '07)*. `https://doi.org/10.1145/1298306.1298314`
[9] Robert Beverly. 2016. Yarrp'ing the Internet: Randomized High-Speed Active Topology Discovery. In *ACM Internet Measurement Conference (IMC)*.
[10] Robert Beverly and Arthur Berger. 2015. Server Siblings: Identifying Shared IPv4/IPv6 Infrastructure via Active Fingerprinting. In *Passive and Active Network Measurement Conference (PAM)*.
[11] Robert Beverly, Ramakrishnan Durairajan, David Plonka, and Justin P. Rohrer. 2018. In the IP of the Beholder: Strategies for Active IPv6 Topology Discovery. In *ACM Internet Measurement Conference (IMC)*.
[12] Kevin Bock, Abdulrahman Alaraj, Yair Fax, Kyle Hurley, Eric Wustrow, and Dave Levin. 2021. Weaponizing Middleboxes for TCP Reflected Amplification. In *USENIX Security Symposium*.
[13] Kevin Borgolte, Shuang Hao, Tobias Fiebig, and Giovanni Vigna. 2018. Enumerating Active IPv6 Hosts for Large-scale Security Scans via DNSSEC-signed Reverse Zones. In *IEEE Symposium on Security and Privacy*.
[14] CAIDA. 2019. The CAIDA UCSD IPv6 Routed /48 Topology Dataset. `https://www.caida.org/data/active/ipv6_routed_48_topology_dataset.xml`.
[15] Costin, Andrei and Zaddach, Jonas and Francillon, Aurélien and Balzarotti, Davide. 2014. A Large-Scale Analysis of the Security of Embedded Firmwares. In *USENIX Security Symposium*.
[16] Tianyu Cui, Gaopeng Gou, Gang Xiong, Chang Liu, Peipei Fu, and Zhen Li. 2021. 6GAN: IPv6 Multi-Pattern Target Generation via Generative Adversarial Nets with Reinforcement Learning. In *IEEE Conference on Computer Communications (INFOCOM)*.
[17] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. 2014. Measuring IPv6 Adoption. *ACM SIGCOMM Computer Communication Review (CCR)* 44, 4 (Aug. 2014).
[18] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. 2014. The Matter of Heartbleed. In *ACM Internet Measurement Conference (IMC)*.
[19] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. ZMap: Fast Internet-wide Scanning and its Security Applications. In *USENIX Security Symposium*.
[20] Asma Enayet and John Heidemann. 2022. Internet Outage Detection Using Passive Analysis. In *ACM Internet Measurement Conference (IMC)*.
[21] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. 2017. Something From Nothing (There): Collecting Global IPv6 Datasets From DNS. In *Passive and Active Network Measurement Conference (PAM)*.
[22] Pawel Foremski, David Plonka, and Arthur Berger. 2016. Entropy/IP: Uncovering Structure in IPv6 Addresses. In *ACM Internet Measurement Conference (IMC)*.
[23] Kensuke Fukuda and John Heidemann. 2018. Who Knocks at the IPv6 Door? Detecting IPv6 Scanning. In *ACM Internet Measurement Conference (IMC)*.
[24] Oliver Gasser, Quirin Scheitle, Pawel Foremski, Qasim Lone, Maciej Korczyński, Stephen D. Strowes, Luuk Hendriks, and Georg Carle. 2018. Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists. In *ACM Internet Measurement Conference (IMC)*.
[25] Oliver Gasser, Quirin Scheitle, Sebastian Gebhard, and Georg Carle. 2016. Scanning the IPv6 Internet: Towards a Comprehensive Hitlist. *CoRR* abs/1607.05179 (2016). arXiv:1607.05179 `http://arxiv.org/abs/1607.05179`
[26] R Gayraud and B Lourdelet. 2010. Network Time Protocol (NTP) Server Option for DHCPv6. RFC 5908. `http://www.ietf.org/rfc/rfc5908.txt`
[27] Google Git. 2016. Android. `https://android.googlesource.com/platform/frameworks/base/+/d3f689bf14a05de735b5cc92dcf20e7226c78690%5E%21/core/res/res/values/config.xml`.

[28] F. Gont. 2014. A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC). RFC 7217 (Proposed Standard). https://doi.org/10.17487/RFC7217

[29] Google. 2023. Geolocation API. https://developers.google.com/maps/documentation/geolocation/overview.

[30] Hang Guo and John Heidemann. 2020. Detecting IoT Devices in the Internet. *IEEE/ACM Transactions on Networking* 28, 5 (Oct. 2020).

[31] John Heidemann, Yuri Pradkin, Ramesh Govindan, Christos Papadopoulos, Genevieve Bartlett, and Joseph Bannister. 2008. Census and Survey of the Visible Internet. In *ACM Internet Measurement Conference (IMC)*.

[32] Bingnan Hou, Zhiping Cai, Kui Wu, Jinshu Su, and Yinqiao Xiong. 2021. 6Hit: A Reinforcement Learning-Based Approach to Target Generation for Internet-Wide IPv6 Scanning. In *IEEE Conference on Computer Communications (INFOCOM)*.

[33] Gokay Huz, Steven Bauer, KC Claffy, and Robert Beverly. 2015. Experience in Using MTurk for Network Measurement. In *ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*.

[34] Young Hyun and k. claffy. 2022. Archipelago Measurement Infrastructure. http://www.caida.org/projects/ark/.

[35] John Kohl, Clifford Neuman, et al. 1993. *The Kerberos network authentication service (V5)*. Technical Report. RFC 1510, september.

[36] Frank Li and David Freeman. 2020. Towards A User-Level Understanding of IPv6 Behavior. In *ACM Internet Measurement Conference (IMC)*.

[37] Zhizhu Liu, Yinqiao Xiong, Xin Liu, Wei Xie, and Peidong Zhu. 2019. 6Tree: Efficient Dynamic Discovery of Active Addresses in the IPv6 Address Space. *Computer Networks* 155 (2019), 31–46.

[38] Matthew Luckie. 2010. Scamper: a Scalable and Extensible Packet Prober for Active Measurement of the Internet. In *ACM Internet Measurement Conference (IMC)*.

[39] Matthew Luckie and Robert Beverly. 2017. The Impact of Router Outages on the AS-level Internet. In *ACM SIGCOMM*.

[40] Aanchal Malhotra, Isaac E. Cohen, Erik Brakke, and Sharon Goldberg. 2016. Attacking the Network Time Protocol. *Network and Distributed System Security Symposium (NDSS)* (2016).

[41] Linda Markowsky and George Markowsky. 2015. Scanning for vulnerable devices in the Internet of Things. In *2015 IEEE 8th International conference on intelligent data acquisition and advanced computing systems: technology and applications (IDAACS)*, Vol. 1.

[42] MaxMind Inc. 2022. MaxMind GeoLite Databases. https://dev.maxmind.com/geoip/geoip2/geolite2/.

[43] Microsoft. 2021. How the Windows Time Service Works. https://docs.microsoft.com/en-us/windows-server/networking/windows-time-service/how-the-windows-time-service-works.

[44] D. Mills. 1985. Network Time Protocol (NTP). RFC 958. http://www.ietf.org/rfc/rfc958.txt

[45] M Morowczynski. 2012. Did your active directory domain time just jump to the year 2000. *Microsoft Server & Tools Blogs http://blogs. technet. com/b/askpfeplat/archive/2012/11/19/did-your-active-directory-domain-time-just-jump-to-the-year-2000. aspx* (2012).

[46] T. Mrugalski, M. Siodelski, B. Volz, A. Yourtchenko, M. Richardson, S. Jiang, T. Lemon, and T. Winters. 2018. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 8415 (Proposed Standard). https://doi.org/10.17487/RFC8415

[47] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. 2017. Target Generation for Internet-Wide IPv6 Scanning. In *ACM Internet Measurement Conference (IMC)*.

[48] Alexander Mylnikov. 2023. Geo-Location API Download Section. https://www.mylnikov.org/download.

[49] Dr. Thomas Narten and Dr. Susan Thomson. 1998. IPv6 Stateless Address Autoconfiguration. RFC 2462. https://doi.org/10.17487/RFC2462

[50] T. Narten and R. Draves. 2001. Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 3041. http://www.ietf.org/rfc/rfc3041.txt

[51] openwifi.su. 2023. OpenWifi.su Dataset. http://openwifi.su/db/.

[52] Ramakrishna Padmanabhan, Amogh Dhamdhere, Emile Aben, kc claffy, and Neil Spring. 2016. Reasons Dynamic Addresses Change. In *ACM Internet Measurement Conference (IMC)*.

[53] Ramakrishna Padmanabhan, Patrick Owen, Aaron Schulman, and Neil Spring. 2015. Timeouts: Beware Surprisingly High Delay. In *ACM Internet Measurement Conference (IMC)*.

[54] Ramakrishna Padmanabhan, Aaron Schulman, Alberto Dainotti, Dave Levin, and Neil Spring. 2019. How to Find Correlated Internet Failures. In *Passive and Active Network Measurement Conference (PAM)*, David Choffnes and Marinho Barcellos (Eds.).

[55] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. 2017. Global Measurement of DNS Manipulation. In *USENIX Security Symposium*.

[56] David Plonka and Arthur Berger. 2015. Temporal and Spatial Classification of Active IPv6 Addresses. In *ACM Internet Measurement Conference (IMC)*.

[57] radiocells.org. 2023. OpenBMap Dataset. https://radiocells.org/.

[58] Philipp Richter, Oliver Gasser, and Arthur Berger. 2022. Illuminating Large-Scale IPv6 Scanning in the Internet. In *ACM Internet Measurement Conference (IMC)*.

[59] Philipp Richter, Ramakrishna Padmanabhan, Neil Spring, Arthur Berger, and David Clark. 2018. Advancing the art of internet edge outage detection. In *ACM Internet Measurement Conference (IMC)*.

[60] Philipp Richter, Florian Wohlfart, Narseo Vallina-Rodriguez, Mark Allman, Randy Bush, Anja Feldmann, Christian Kreibich, Nicholas Weaver, and Vern Paxson. 2016. A Multi-perspective Analysis of Carrier-Grade NAT Deployment. In *ACM Internet Measurement Conference (IMC)*.

[61] RIPE. 2017. Best Current Operational Practice for Operators: IPv6 Prefix Assignment for End-Users - Persistent vs Non-Persistent, and What Size to Choose. https://www.ripe.net/publications/docs/ripe-690.

[62] Justin P. Rohrer, Blake LaFever, and Robert Beverly. 2016. Empirical Study of Router IPv6 Interface Address Distributions. *IEEE Internet Computing* (Aug. 2016).

[63] Erik Rye and Robert Beverly. 2023. IPvSeeYou: Exploiting Leaked Identifiers in IPv6 for Street-Level Geolocation. In *IEEE Symposium on Security and Privacy*.

[64] Erik Rye, Robert Beverly, and kc claffy. 2021. Follow the Scent: Defeating IPv6 Prefix Rotation Privacy. In *ACM Internet Measurement Conference (IMC)*.

[65] Erik C Rye and Robert Beverly. 2020. Discovering the IPv6 Network Periphery. In *Passive and Active Network Measurement Conference (PAM)*.

[66] Said Jawad Saidi, Oliver Gasser, and Georgios Smaragdakis. 2022. One Bad Apple Can Spoil Your IPv6 Privacy. *ACM SIGCOMM Computer Communication Review* 52, 2 (2022).

[67] Aaron Schulman and Neil Spring. 2011. Pingin'in the rain. In *ACM Internet Measurement Conference (IMC)*.

[68] Lion Steger, Liming Kuang, Johannes Zirngibl, Georg Carle, and Oliver Gasser. 2023. Target Acquired? Evaluating Target Generation Algorithms for IPv6. In *Network Traffic Measurement and Analysis*.

[69] Stephen D Strowes. 2017. Bootstrapping active IPv6 measurement with IPv4 and public DNS. *arXiv preprint arXiv:1710.08536* (2017).

[70] tumi8. 2022. ZMapv6: Internet Scanner with IPv6 Capabilities. https://github.com/tumi8/zmap.

[71] WiGLE – All the Networks. Found by Everyone. 2023. WiGLE – All the Networks. Found by Everyone. https://wigle.net.

[72] Tao Yang, Zhiping Cai, Bingnan Hou, and Tongqing Zhou. 2022. 6Forest: An Ensemble Learning-Based Approach to Target Generation for Internet-Wide IPv6 Scanning. In *IEEE Conference on Computer Communications (INFOCOM)*.

[73] Sebastian Zander, Lachlan LH Andrew, and Grenville Armitage. 2014. Capturing Ghosts: Predicting the Used IPv4 Space by Inferring Unobserved Addresses. In *ACM Internet Measurement Conference (IMC)*.

[74] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitraş, Alan Mislove, Aaron Schulman, and Christo Wilson. 2014. Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed. In *ACM Internet Measurement Conference (IMC)*.

[75] Johannes Zirngibl, Lion Steger, Patrick Sattler, Oliver Gasser, and Georg Carle. 2022. Rusty Clusters? Dusting an IPv6 Research Foundation. In *ACM Internet Measurement Conference (IMC)*.

[76] Maya Ziv, Liz Izhikevich, Kimberly Ruth, Katherine Izhikevich, and Zakir Durumeric. 2021. ASdb: A System for Classifying Owners of Autonomous Systems. In *ACM Internet Measurement Conference (IMC)*.

Sammy: smoothing video traffic to be a friendly internet neighbor

Bruce Spang
Stanford University

Shravya Kunamalla
Netflix

Renata Teixeira
Netflix

Te-Yuan Huang
Netflix

Grenville Armitage
Netflix

Ramesh Johari
Stanford University

Nick McKeown
Stanford University

ABSTRACT

On-demand streaming video traffic is managed by an adaptive bitrate (ABR) algorithm whose job is to optimize quality of experience (QoE) for a single video session. ABR algorithms leave the question of sharing network resources up to transport-layer algorithms. We observe that as the internet gets faster relative to video streaming rates, this delegation of responsibility gives video traffic a burstier on-off traffic pattern. In this paper, we show we can substantially smooth video traffic to improve its interactions with the rest of the internet, while maintaining the same or better QoE for streaming video. We smooth video traffic with two design principles: application-informed pacing, which allows ABR algorithms to set an upper limit on packet-by-packet throughput, and by designing ABR algorithms that work with pacing. We propose a joint ABR and rate-control scheme, called Sammy, which selects both video quality and pacing rates. We implement our scheme and evaluate it at a large video streaming service. Our approach smooths video, making it a more friendly neighbor to other internet applications. One surprising result is that being friendlier requires no compromise for the video traffic: in large scale, production experiments, Sammy improves video QoE over an existing, extensively tested and tuned production ABR algorithm.

CCS CONCEPTS

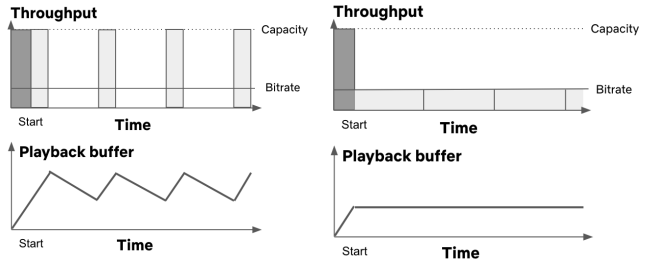
• **Networks** → **Cross-layer protocols**; **Network resources allocation**; • **Information systems** → **Multimedia streaming**;

KEYWORDS

Video streaming; Adaptive bitrate algorithms; Congestion control algorithms; Network friendliness

1 INTRODUCTION

On-demand streaming video traffic from services like Netflix and YouTube currently comprises 60-75% of internet traffic [57]. This fraction is likely even higher during peak viewing hours. With a single application having such a large volume of traffic, as a community we should make it as good a neighbor as possible. If we do so, we will improve the internet for all applications that share it.



(a) Video traffic today. **(b) Smoother, same QoE.**

Figure 1: A few seconds of a typical streaming video session. Today, video has on periods (light grey) where it sends data as fast as the network supports (a). But as shown in (b), we can smooth throughput and reduce congestion without impacting video QoE.

Streaming video traffic today has an on-off, bursty traffic pattern shown in Figure 1a: every few seconds, video switches between an on period of sending data as fast as possible and an off period of silence. This is a well known phenomenon [54], and arises from the standard architecture used to stream video. Videos are split into "chunks" of a few seconds each and stored on a server. Server-side congestion control algorithms send chunks to the video player as fast as the network allows, creating the on periods shown in Figure 1a. The video player then puts the chunks into a playback buffer, to be played back as needed. Each chunk is encoded at a number of different bitrates: from a higher quality, larger chunk-size, to a lower quality, smaller chunk-size. An adaptive bitrate (ABR) algorithm selects the bitrate of each chunk. When network bandwidth is higher than the bitrate of the chunk, chunks arrive at the client faster than they are played back and so the playback buffer grows. Client buffers can only store a limited number of chunks, so ABR algorithms periodically pause to make room for new chunks creating the off periods in Figure 1a. For more background, see Section 2.

Over the past decade, video traffic has been getting more bursty: on periods are getting shorter, and off periods are getting longer. A decade ago, home access speeds and video bitrates were both on the order of a few megabits per second [15, 38], and so video traffic

ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3604839

spent most of its time in on periods [54]. Since then, median last-mile network data-rates to the home have improved, worldwide, to tens or hundreds of megabits per second [16, 63]. Video encoding has also improved: today a 1080p video requires only a few megabits per second [38] and a 4k video requires typically less than twenty megabits per second [45]. Videos download faster, playback buffers fill up faster, and video traffic has shorter, faster on periods.[1]

Video traffic can be smoother. Figure 1b shows the exact same video streaming trace, smoothed out. We have reduced the *chunk throughput* (the throughput during on periods) to the level of the video bitrate—well below the network capacity. From the end user's perspective, the quality of experience (QoE) of the video is identical. Video QoE is measured by *video quality* (how good the video looks), *play delay* (how long the video takes to start playing), and *rebuffers* (times when the video playback is interrupted because data is not available). Both sessions in Figure 1 have the same QoE: quality is the same (the same number of bytes are downloaded in the same amount of time), the play delay is the same (the width of the dark gray box is the same), and there are no rebuffers (the buffer never goes to zero).

Smoothing video traffic as in Figure 1b has benefits to neighboring traffic sharing the same network. There are well-studied consequences to congestion control sending as fast as possible during on periods, including increased packet loss and queueing delay, bufferbloat [24], and unfairness between flows [1, 7, 8, 11, 12, 18, 32, 33, 35, 36, 42, 60, 68–71]. Conventional wisdom in queueing theory also suggests that burstiness increases router queues and network congestion [27], and so is detrimental to neighboring traffic.

By reducing chunk throughput *below* the capacity in Figure 1a, we avoid these issues completely. There will be *no* queueing delay or packet loss. A short HTTP request issued during an on period could complete faster with more available bandwidth and lower queueing delay. A longer-lived video conferencing flow would see more consistent throughput and delay. Reducing burstiness should benefit everyone.

The major challenge of making video traffic smoother at scale is doing so without making video traffic perform worse. Video QoE is important for the experience of the people watching the video [17, 40, 75] and we would not want to smooth video traffic—the majority use of the internet—by making its users suffer. There are two different aspects of this challenge.

First, there is a fundamental limit to how much traffic can be smoothed without impacting QoE. For example, we could smooth Figure 1b even more by reducing the throughput before playback starts to match the rest of the session. But this would increase play delay. No buffer will be built up, so the playback will rebuffer if throughput varies.

Second, ABR algorithms have historically had a core assumption that measurements of chunk throughput give them accurate estimates of the available bandwidth of the network [4, 31, 35, 59, 64–66, 72, 73]. Reducing chunk throughput breaks this assumption, and could cause an ABR algorithm to select lower qualities—artificially lowering QoE.

[1]Anecdotally, the median Netflix session today has an average throughput 13x higher than its average bitrate.

In this paper, we present a novel solution to this challenge and make video traffic smoother. Remarkably, our approach appears to benefit everyone: we are able to substantially smooth video traffic while slightly *improving* QoE relative to existing, finely-tuned production video streaming systems.

To smooth video traffic, we allow ABR algorithms to directly limit the packet-by-packet sending rate using a new technique called *application-informed pacing*. An ABR algorithm might ask for a chunk of video to be delivered at no more than one packet per millisecond, and a congestion control algorithm sends packets no faster than once every millisecond using TCP Pacing [1, 13, 26, 28, 47, 56, 71]. This allows the ABR algorithm to smooth out throughput across the full range of timescales: from the level of a few packets, to an entire chunk, to an entire video session. Application-informed pacing is described in more detail in Section 3.2.

We next propose *Sammy*,[2] an algorithm that selects both bitrates and pacing rates to achieve high video QoE and improve smoothness. Sammy is described in Section 4. Our key insight, described in Section 3.1, is that while ABR algorithms have historically *used* measurements of available bandwidth to make their decisions, they do not *need* accurate estimates to achieve good QoE.

We implement Sammy at Netflix and evaluate it with large scale, production experiments in Section 5. Sammy substantially smooths video traffic: reducing chunk throughput to roughly three times higher than the highest bitrate of a video. In production experiments, this lowers chunk throughput by 61% at the median. This improves congestion metrics: improving retransmissions by 35%, and RTTs by 14%. Surprisingly, Sammy actually *slightly improves* video QoE relative to production values despite this reduction in throughput: improving initial video quality by 0.2%, overall quality by 0.03%, and play delay by 1.3% while maintaining rebuffers.

We present illustrative lab experiments in Section 6 in which Sammy improves the performance of neighboring traffic by increasing its throughput and reducing queueing delay. Sammy improves delay for a neighboring UDP flow by 51%, improves throughput for a TCP flow by 28%, improves response times for HTTP traffic by 18%, and improves play delay for another video session by 4%.

Streaming video services are incentivized to deploy Sammy. First, neighboring traffic could easily be from the same video service. By improving performance for its neighbors, Sammy improves performance for the video service itself. Second, Sammy forces an ABR algorithm to be careful about its use of throughput estimates. This exercise gave us a slight QoE improvement, and could yield larger improvements for other ABR algorithms.

This work is a first step towards smoothing video traffic. We conclude in Section 7 by highlighting that there is still more work to be done. As a community, we have an opportunity to further smooth traffic, video and beyond. After all, a smoother internet benefits everyone.

To demonstrate the deployability of Sammy, we have released an open source prototype [61] which uses off the shelf components including an unmodified dash.js player and the Fastly CDN.

Ethical considerations: Our experiments involve live traffic running on a large video streaming service. Sammy makes video traffic friendlier to its neighbors while improving QoE, so we believe

[2]As of this writing, Sammy is the current reigning world's fastest snail [53].

our experiments are beneficial. Netflix regularly runs rigorous A/B testing for every change it makes to its service. Its customers have the ability to opt out of experiments, if they choose to.

2 BACKGROUND AND RELATED WORK

The burstiness of video traffic arises from the standard architecture of modern video streaming services. One of the core design principles of the internet is layering [14], conceptually separating applications from underlying transport protocols like TCP or QUIC. The internet community has kept a minimalist inter-layer interface, giving applications little ability to express their service goals. As a result, transport protocols optimize for transport-level goals: maximizing throughput, avoiding congestion, and splitting throughput fairly. Adaptive bitrate (ABR) algorithms select bitrates to ensure that viewers get the best quality of experience (QoE) possible, given whatever throughput is picked by the congestion control algorithm.

In this section, we will give an overview of existing work and describe why this architecture makes it difficult to achieve our goal of smoothing video traffic while achieving high QoE. In this section we focus on the most relevant papers. For a more complete overview, there are a number of wider surveys [41, 52, 58].

2.1 ABR algorithms

When a network has limited bandwidth, there is a tradeoff between the three major video QoE metrics: quality, startplay delay, and rebuffers. A video playback can have both high quality and no rebuffers if it incurs a high play delay by downloading the entire video before it starts. It can start quickly in a slow network by either picking a low quality or rebuffering after playback starts. The role of an ABR algorithm is to manage this tradeoff between the different QoE metrics, and ensure that the user gets the best possible QoE. Note that this does not necessarily mean picking the highest video quality—in a given network, a very low quality might significantly reduce rebuffers and have the best QoE.

Videos are split into chunks of a few seconds each. Each chunk is encoded in a ladder of different bitrates: from a small, low-quality version to a larger, high-quality version. A video provider will allow a particular device in a particular network to use some subset of this ladder based on the user's plan, device limitations, and other business policies. The ABR algorithm chooses a rung from this ladder for each chunk. The transport layer then splits that chunk into packets and sends each packet to the video client. When chunks are downloaded, they are added to a playback buffer in the video client. Even if the network is unavailable, the client can continue playing as long as there are chunks in the buffer.

When it takes too long to download a chunk, the buffer can shrink and it may be impossible to maintain high video quality without rebuffers. To deal with this, ABR algorithms can pick lower bitrates to grow the buffer and avoid rebuffers. There are two main types of ABR algorithms in use today:

Throughput-based: An ABR algorithm that takes explicit throughput measurements from the network, and uses them to select bitrates [4, 35, 49, 59, 64, 66, 72, 73]. Typical algorithms produce some estimate based on chunk throughput, and then use it to optimize the various QoE metrics. ABR algorithms can also use throughput in other ways, for example, Oboe [4] switches between several parameter settings based on throughput measurements. VOXEL makes modifications to the transport layer to drop video frames in challenging network conditions. It is generally understood [72, 73] that throughput-based algorithms will perform better the more accurately they are able to predict throughput of upcoming chunks.

Buffer-based: An ABR algorithm may select a bitrate based only on the buffer level [31, 65]. When the buffer is low, the algorithm will pick the lowest bitrate. When the buffer is high, it will pick the highest bitrate. Over time, these algorithms converge to an average bitrate close to the average chunk throughput. In effect, the buffer size encodes the past available bandwidth measurements from the network. In practice, buffer-based algorithms can also include a throughput-based component during startup [64].

Existing ABR algorithms rely on the available bandwidth measurements produced by congestion control algorithms. By decreasing chunk throughput, we could make existing ABR algorithms perform worse. We discuss how we address this in Section 3.2.

Until now, ABR algorithms have focused on maximizing the quality of experience (QoE) for a video streaming client given whatever throughput is chosen by congestion control algorithms. ABR algorithms do not make choices about throughput. In contrast, our goal is to design an algorithm that smooths video traffic and achieves high QoE while improving the internet for neighboring traffic.

2.2 Congestion control

Once an ABR algorithm has selected a chunk, it is the job of congestion control algorithms to decide how fast to send the packets of that chunk into the network. Congestion control algorithms balance competing goals: achieving high throughput, avoiding congestion, and fairly splitting network resources among users [50, Sec. 3.2].

There is a long line of research on congestion control, and we refer the reader to existing surveys for details [52]. It is challenging (if not impossible [74, 76]) to simultaneously achieve all the goals of congestion control, and there are examples of congestion control algorithms struggling with packet loss and queueing delay, bufferbloat [24], and unfairness [1, 7, 8, 11, 12, 18, 32, 33, 35, 36, 42, 60, 68–71]. By focusing on the needs of video QoE, Sammy gives up the goal of achieving high chunk throughput and so is able to improve congestion and leave more bandwidth available for other users of the network.

Our work uses the classic idea of TCP Pacing: a mechanism for adding delay between successive packet sends to reduce the size of bursts, and reduce packet drops and queueing delay [1, 13, 26, 28, 47, 56, 71]. Pacing gives packets a constant interarrival time, which theoretically minimizes queueing delay in general settings [27]. In practice, pacing reduces congestion [1, 47, 71]. Typically the time between successive packets is set to a value larger than the cwnd/RTT [1, 67], which reduces burstiness at the packet-level, but does not reduce chunk throughput. BBR [13] is a congestion control algorithm that directly adjusts the pace rate, but it aims to pace close to the bottleneck capacity while Sammy aims to pace significantly lower.

There has been prior work on congestion control to improve fairness among competing video clients. There are "scavenger" congestion control algorithms like LEDBAT [55] and PCC Proteus [46], that give up throughput when competing with a non-scavenger

Paper	Main Goal	Smoothing Mechanism	Eval.	Preserves QoE?	Smooths below avail. bandwidth?
Our work	Smoothness	ABR-informed Pacing	Prod.	✓	✓
TCP Pacing [1]	Packet bursts	Pacing	Prod.	✓	✗
Trickle baseline [25]	Wasted buffer	Token Bucket	?	?	✓
Trickle [25]	Packet bursts	CWND limit	Prod.	?	✓
[59]	Wasted buffer	Delays TTFB	Lab	✗	✗
SABRE #1 [44]	Bufferbloat	RWND limit	Lab	✗	✓
SABRE #2 [44]	Bufferbloat	RWND limit	Lab	✗	✓
[3]	Video fairness	Token Bucket	Lab	✗	✓
[9]	Video fairness	Token Bucket	Lab	✗	✗

Table 1: Related work on smoothing video traffic either does not preserve QoE or does not reduce throughput below the available network bandwidth.

congestion control algorithm. These were originally designed to reduce the impact of BitTorrent traffic on other internet traffic [55], but the PCC Proteus [46] authors describe a hybrid mode, in which video traffic switches from non-scavenger to scavenger mode when the throughput exceeds a threshold. The authors show that this improves performance when multiple video clients compete. Minerva [48] improves fairness between competing video sessions by sharing proportionally based on a measure of perceptual quality. These approaches will fully utilize the network when no neighboring traffic is present. In contrast, Sammy does not focus on fairness and instead consistently sends at a rate closer to the video bitrate. Surprisingly, we show that consistently smoothing (even when neighboring traffic is not present) achieves similar performance to fully utilizing the network.

2.3 Reducing Burstiness for Video Traffic

There has been prior work on reducing the burstiness of video traffic. The novelty of our work is that by designing a joint ABR and rate limiting algorithm, we are able to reduce chunk throughput below the available bandwidth of a network *while achieving comparable QoE to today's top ABR algorithms.*

Related work on reducing video burstiness is summarized in Table 1. There are a number of differences to our work. The baseline algorithm described in Trickle [25] reduces chunk throughput based on the video encoding rate with the goal of reducing wasted buffers when videos end early. The impact of this algorithm on QoE, smoothness, or neighboring traffic is not evaluated in the paper but since algorithm reduces buffer sizes it likely has some impact on QoE. In contrast, our work explores how to design an ABR algorithm to improve smoothness while achieving high QoE. Work on pacing like TCP Pacing [1] and Trickle (relative to their baseline) [25] focuses on reducing per-packet burstiness while maintaining congestion control-selected throughput. There is work [3, 44, 59] which reduces throughput below a congestion control algorithm's selected rate using mechanisms other than pacing, but this work does not preserve video QoE (typically because it treats ABR algorithms as a black box). Finally, there is some related work [9, 43] which delays the time to first byte (TTFB) of the HTTP response. This reduces buffer sizes, but does not improve smoothness over typical congestion control algorithms.

There has also been prior measurement work, observing that some video traffic reduces burstiness in practice using some of the mechanisms described above [5, 54].

Our work has some similarity to real-time video streaming systems (like FaceTime and Zoom), that are designed differently than on-demand systems (like Netflix and YouTube). These systems also need to pick bitrates and sending rates for videos. To pick bitrates, real-time systems pick an encoding bitrate for each frame, while on-demand systems pick bitrates from an offline-chosen ladder. Real-time systems are built on top of UDP and so need to decide when to send each packet, while on-demand systems have historically relied on congestion control algorithms. Because of these differences, real-time systems have less of a distinction between congestion control and bitrate adaptation schemes. For instance, the Google Congestion Control algorithm [29] for WebRTC picks sending rates using a delay-based algorithm and aims to match the encoding bitrate to the sending rate. Salsify [23] relies on packet-pair techniques, adjusts its sending rates to control queuing delay, and picks bitrates based on the chosen sending rates. In contrast, our work focuses on making *on-demand* systems friendlier while achieving comparable QoE to today's top on-demand systems.

2.3.1 The challenge of reducing burstiness with existing ABR algorithms. All prior work on reducing burstiness for video traffic falls short of achieving high video QoE because they treat ABR algorithms as a black box.

Today's ABR algorithms are designed to use either explicit measurements of available bandwidth (as in the case of throughput-based ABR algorithms) or implicit ones collected through the accumulation of a buffer (as with a buffer-based algorithm). If we reduce burstiness by reducing throughput, this changes available bandwidth and can easily cause ABR algorithms to select lower bitrates and reduce QoE.

As an example of how reducing chunk throughput can cause QoE issues, imagine a simple ABR algorithm which measures the minimum throughput x over the last few chunks and picks the highest video bitrate $< cx$ for some constant c.[3] Say we set $c = 0.5$, and picked a pacing rate of 1.5x the video bitrate. This would cause a downward spiral [30]: we would start with video bitrate B, pick a pacing rate of $1.5 \cdot B$, and measure a throughput of $x = 1.5 \cdot B$. We would then pick the highest video bitrate lower than $0.5 \cdot (1.5 \cdot B) =$

[3]This is the default dash.js algorithm when the buffer is low [64].

$0.75 \cdot B$, and would switch down. We would continue switching down until we reached the lowest bitrate. This example shows that we cannot treat ABR algorithms as a black box when reducing burstiness and maintain the same QoE.

As another example, imagine all chunks are one second long, and we pick a pace rate equal to the chunk bitrate. The chunk will download in exactly the same amount of time as it takes to play, one second. As a result, the playback buffer will not increase and remain at a near-zero level. If chunk throughput varied at all, there would be a rebuffer. For buffer-based algorithms, this also means that the buffer will not grow large enough to select a high bitrate chunk, reducing video quality. Note that picking a pace rate equal to or even lower than the chunk bitrate may be an effective approach. If the video buffer is relatively large, it may be desirable to give up some buffer growth to improve smoothness. By pacing at the chunk bitrate, the buffer can be kept at the same relatively high level. By pacing slightly below the chunk bitrate, video traffic can be made even smoother while only slightly shrinking the buffer.

3 DESIGN DISCUSSION

In this section, we will describe the key components of Sammy: the key idea behind designing an ABR algorithm for pacing, and the application-informed pacing mechanism Sammy uses to limit throughput. In Section 4, we will use these components when introducing Sammy.

3.1 ABR algorithms with paced throughput

Application-informed pacing reduces throughput below the available bandwidth of the network. With pacing, an ABR algorithm might *never* learn the available bandwidth of a network. As discussed in Section 2, existing ABR algorithms rely on measurements of available bandwidth. So how can an ABR algorithm optimize QoE with pacing?

The main idea behind our approach is that while ABR algorithms have historically *used* estimates of available bandwidth to make their decisions, they do not *need* accurate available bandwidth estimates to achieve good QoE. An ABR algorithm only needs to know whether the network can support the highest bitrate, or if it needs to reduce video quality to improve QoE.

For example, imagine streaming a video with a top bitrate of 10 Mbps. Once we have built up a small playback buffer, the ABR algorithm only needs to know whether the network throughput is high enough to sustain the top bitrate without rebuffering. If the available bandwidth is 100 Mbps or 1000 Mbps, a good algorithm would still pick the top bitrate.

Instead of relying on accurate estimates of available bandwidth, Sammy can use a pacing-informed ABR algorithm that instead solves a decision problem: *is the available bandwidth of the network enough to pick a bitrate, or not?* As we discuss in Section 4.2, many commonly used ABR algorithms implicitly rely on such a decision problem and do not need accurate estimates of available bandwidth.

An alternate approach would be to estimate available bandwidth despite pacing, for instance using packet pair techniques [37, 39], or not pacing some portion of requests. We did not pursue this, in favor of an approach that avoids exact throughput estimation in the first place.

3.2 Limiting throughput with application-informed pacing

Streaming video traffic is bursty because congestion control algorithms send as fast as the available network bandwidth, which is often higher than needed for good QoE. Our approach is to have the ABR algorithm reduce the server's sending rate with *application-informed pacing*: a new technique that allows applications to set an upper limit on the server's sending rate. By carefully limiting bursts, an ABR algorithm can reduce chunk throughput below the available network bandwidth while achieving high QoE.

In application-informed pacing, the ABR algorithm selects a pace rate and sends this rate to the server via an HTTP header. The server uses TCP Pacing as described in Section 2 and [1, 28, 71] to limit the sending rate at the server side. To achieve a desired rate of R packets per second, the server delays sending packets to ensure that there is a delay of at least $1/R$ seconds between the starts of successive packets.

Application-informed pacing runs in combination with a congestion control algorithm, and the pace rate is an upper limit on the sending rate. Congestion control algorithms can still limit the sending rate by reducing the congestion window or pace rate. If an application requests a pace rate higher than network bandwidth, congestion control algorithms will operate as normal and pick a lower sending rate. Because the resulting throughput is *at most* the requested pace rate, Application-informed pacing is TCP-fair to existing internet traffic.

Deployability. Application-informed pacing is readily deployable. TCP Pacing is already part of the Linux kernel [19], is used in production at Google [13, 47, 56], and is available in certain NICs [56]. In Linux, an HTTP server can implement application-informed pacing by setting the `SO_MAX_PACING_RATE` socket option [20] to an application-provided value.

There is CDN support for application-informed pacing. Akamai supports CMCD, a video standard that allows clients to limit server-side throughput using the `rtp` parameter [2, 6]. Fastly allows setting TCP pace rates based on the value of an HTTP header [22].

Application-informed pacing is an example of cross-layer design, and there are lots of other ways to limit a server's throughput. A system could use client receive windows to limit throughput [44], could limit a server's congestion window [25], or could use a server-side token bucket to reduce rates [3]. These techniques might be more bursty than application-informed pacing, but might be more easily deployable in certain settings.

4 SAMMY

We will now describe Sammy, our system that jointly selects bitrates and pace rates to smooth out video traffic while ensuring high QoE. In a significant shift from conventional video streaming systems, Sammy's primary mechanism for throughput selection is pace rate selection by the ABR algorithm, with congestion control acting as a backup to ensure TCP-fairness to existing systems. Sammy selects pace rates using information from the ABR algorithm, like buffer level and player state. To select video bitrates, Sammy relies on a pacing-aware ABR algorithm which ensures high QoE even without accurate estimates of available bandwidth. One of our

main contributions is showing that a wide range of existing ABR algorithms *already* do not require precise estimates of available bandwidth for high QoE.

Sammy is divided into two distinct algorithms: one for the initial phase (before playback starts), and one for the playing phase. This division follows naturally from the different QoE goals of each phase. During the initial phase, it is important to start playback quickly. After playback starts, there can be no change to play delay and the QoE goal shifts towards avoiding rebuffers with high quality.

4.1 Algorithm for the Initial Phase

The initial phase of a video is the time period between when a user initiates playback and the playback actually starts. During this phase, ABR algorithms download chunks to build up a small buffer before beginning playback. Sammy has four competing QoE goals in this phase:

(1) **High initial quality:** Sammy should pick high bitrates for the first few chunks of video playback.
(2) **Few rebuffers:** Sammy should build up a sufficiently large buffer before playback starts to avoid rebuffers.
(3) **Low play delay:** Sammy should begin playback as quickly as possible.
(4) **Smoothness:** Sammy should smooth out traffic by picking low pace rates.

In the initial phase, we will not aim to improve smoothness and will allow conventional congestion control algorithms to pick chunk throughput. If we reduce chunk throughput with the same initial quality and starting buffer size, we will be downloading the same number of bytes in a longer period of time and potentially increase play delay. The initial phase is a small fraction of traffic (typically a few seconds over a tens of minutes long session), so not pacing has a minor impact on overall smoothness.

The challenge in the initial phase is making bitrate selections with relatively few throughput measurements. ABR algorithms typically deal with this challenge by using historical throughput from the playing phase of previous sessions [34, 66]. But if Sammy reduces chunk throughput in the playing phase of previous sessions, this can change these estimates and result in lower initial quality (as shown in experiments in Section 5.5).

In the initial phase, Sammy requires an ABR algorithm whose initial bitrate selections are not affected by the throughput from the playing phase of other sessions. This can be accomplished in many ways. For an existing ABR algorithm that uses historical throughput estimates, we add separate *initial* throughput estimates and update these estimates only with throughput from the initial phase. For separate systems that predict initial throughput like CS2P [66], this can be done by supplying this system *only* with initial throughput measurements. Other ABR algorithm may need no modification, for instance an ABR algorithm which always selects the lowest quality for the first chunk, or Puffer [72] which uses statistics about the establishment of a TCP connection to estimate initial throughput.

In our experiments in Section 5, we record historical throughput measurements from *only* the initial phases of previous sessions on the same device and use them to select the initial bitrate. Sammy uses these estimates with Netflix's existing bitrate selection algorithm for the initial phase. This is described in pseudocode in Algorithm 1.

4.2 Algorithm for the Playing phase

During the playing phase, Sammy selects both a bitrate and a pace rate to balance three competing QoE goals:

(1) **High quality:** Sammy should pick high video bitrates.
(2) **Few rebuffers:** Sammy should avoid playback interruptions by keeping the buffer above zero.
(3) **Smoothness:** Sammy should smooth out traffic by picking low pace rates.

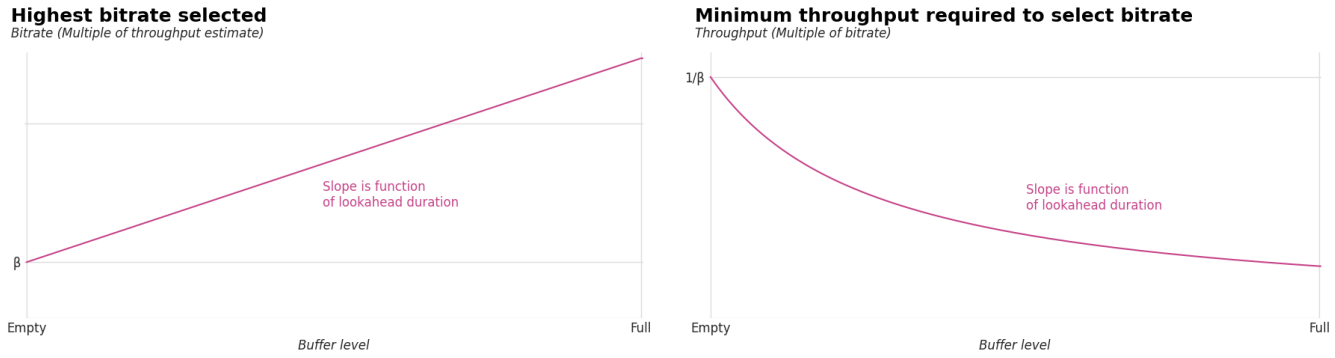
Picking a lower pace rate can affect all three goals: it improves smoothness, reduces throughput estimates (potentially impacting video quality), and causes buffers to grow more slowly (potentially impacting rebuffers).

4.2.1 Sammy's conceptual design. Sammy includes two main components in the playing phase: an ABR algorithm and a pace rate selection algorithm. Our overall strategy for the playing phase will be to take a given ABR algorithm and reduce chunk throughputs as much as possible without impacting bitrate selection. If an ABR algorithm picks the same sequence of bitrates with and without pacing, Sammy will achieve the same video quality with and without pacing. Achieving the same *QoE* with and without pacing is then just a matter of ensuring that the buffer is large enough to prevent rebuffers.

Instead of proposing a single new ABR algorithm, we will describe how to analyze a class of *pacing-aware* ABR algorithms to understand how much throughput can be reduced without impacting QoE. We then use a buffer-based algorithm [31] to pick high pace rates when the buffer is low (growing the buffer more quickly) and lower pace rates when the buffer is high (growing the buffer more slowly), while ensuring that the pace rates stay above the minimum required throughput from our analysis. This approach ensures that Sammy is easily deployable in existing, large-scale video streaming services. Surprisingly, we show that throughput can be significantly lower without impacting QoE for existing ABR algorithms.

Pacing-aware ABR algorithms: As discussed in Section 3.1, instead of relying on an ABR algorithm which *accurately estimates* available bandwidth, Sammy will use a pacing-aware ABR algorithm that relies on a decision problem: *is the available bandwidth high enough to pick a bitrate, or not?* This decision problem gives us a threshold throughput—a minimum value of the algorithm's throughput estimate that will cause it to pick the same bitrate. This threshold gives Sammy room to decrease throughput via pacing. As long as throughput estimates stay above this threshold, Sammy can decrease chunk throughput without changing bitrate decisions.

Fortunately, many existing ABR algorithms already implicitly use such a decision problem. As an example, consider a typical throughput-based algorithm: the HYB algorithm [4], modified to use lookahead (i.e. take upcoming chunk sizes into consideration). This analysis also applies to MPC algorithms [73] with appropriately chosen utility functions, ABR algorithms without lookahead, and so on. The HYB algorithm computes a throughput estimate from recent throughput measurements, and multiplies this estimate by



(a) HYB picks higher bitrates as the buffer grows.

(b) HYB has a minimum throughput needed to select a chunk.

Figure 2: By analyzing how an example throughput-based ABR algorithm (HYB) picks bitrates as a function of chunk throughput estimates (a), we can find a lower bound on pace rates to avoid impacting QoE (b). For example, to pick a bitrate with an empty buffer, HYB requires a throughput at least $1/\beta$ times higher than that bitrate.

a parameter $\beta \in [0, 1]$ to offset prediction errors. It then uses a standard buffer update equation [30] to predict how the buffer evolves over the lookahead duration. It picks the highest bitrate which keeps the buffer above zero.

To better understand the behavior of this algorithm, in Appendix A we analyze how the playback buffer evolves over time. Let D_T be the lookahead duration of the upcoming T chunks. We show that for a throughput x, bitrate r, and starting buffer size B_0, the buffer evolves according to

$$B_T = B_0 + D_T - D_T \frac{r}{\beta x}.$$

HYB picks the highest bitrate which keeps $B_T > 0$, which gives us the following constraint on the bitrate r:

$$r \leq \beta x \left(1 + \frac{B_0}{D_T}\right).$$

This function is shown in Figure 2a: as buffer size and throughput grows, HYB will pick higher bitrates[4].

As a corollary, this gives us a minimum throughput required to pick a bitrate r.

$$x \geq r\beta^{-1} \left(1 + \frac{B_0}{D_T}\right)^{-1}. \quad (1)$$

We graph this function in Figure 2b: when the buffer is empty, HYB needs an estimate of throughput equal to the bitrate divided by β. For example, if $\beta = 0.5$ and the buffer is empty, HYB will pick a bitrate provided the throughput is at least twice the bitrate. When the buffer is lower, HYB can select a bitrate with a lower throughput.

Equation 1 is the implicit function HYB uses to decide whether or not throughput is high enough to select a bitrate. In order to avoid impacting bitrate selection, we must pick a pace rate higher than this value. When the buffer is empty, we must pick a pace rate of at least $1/\beta$ times the top bitrate. When the buffer is larger, we can pick a lower pace rate without impacting bitrate selection.

[4]Previous research on ABR (e.g. [31]) has made a distinction between throughput-based and buffer-based algorithms. Interestingly, this analysis shows that while the description of HYB seems to be a classic throughput-based algorithm, implicitly it uses a buffer-based approach to select bitrates.

Sammy's pace-rate selection. Videos are encoded into a ladder of bitrates. Sammy takes the highest bitrate in this ladder, call this value r. When the buffer is empty, Sammy paces at a multiple of highest bitrate, e.g. at a rate of $c_0 \cdot r$ for some constant c_0. When the buffer is full, Sammy paces at a different multiple of the highest bitrate, e.g. at a rate of $c_1 \cdot r$ for some constant c_1. We set the parameters c_0, c_1 so that the resulting pace rate is always above the minimum throughput required to pick the highest bitrate given by Equation (1) and Figure 2. We can choose higher parameter values than this to tune the tradeoff between rebuffers and pace rates. Once Sammy selects a pace rate, it communicates this rate to the transport layer using application-informed pacing, as discussed in Section 3.2.

4.3 Sammy's implementation

In the first part of the section, we have presented a more generic version of Sammy that works with a variety of pacing-aware ABR algorithms. Here we describe the specific implementation of Sammy we use for experiments in Section 5. Sammy uses Netflix's production ABR algorithm, which is an MPC-style algorithm. This is a proprietary algorithm and we cannot describe it in detail.

During the playing phase, we use Netflix's production ABR algorithm without modification. During the initial phase, we record a separate set of historical initial throughput measurements and use these measurements in place of Netflix's existing historical throughput measurements. The distribution of initial throughput is slightly different than Netflix's existing measurements, and so accordingly we retune Netflix's initial bitrate selection logic to use these measurements without decreasing QoE. We present the results of experiments with just these changes in Section 5.7.

Algorithm 1 summarizes Sammy's implementation. To demonstrate the deployability of Sammy and show how its different components work in practice, we have released an open source prototype [61] of Sammy's playing phase. Our prototype uses off the shelf components including an unmodified dash.js player and the Fastly CDN. Our prototype likely decreases QoE relative to the production dash.js implementation (e.g. the parameters are untuned and we

make no modifications to dash.js's initial bitrate selection), and we leave achieving QoE parity as an exercise to the interested reader.

Algorithm 1 Sammy's bitrate and pace rate selection

Require: ABR algorithm, parameters $c_0, c_1 \geq 0$
  if ABR is in initial phase **then**
    bitrate $\leftarrow$ ABR select(initial throughput measurements)
    pace rate $\leftarrow$ no pacing
  else
    bitrate $\leftarrow$ ABR select(all throughput measurements)
    $B \leftarrow$ buffer/max buffer
    multiplier $\leftarrow c_1 \cdot B + c_0 \cdot (1 - B)$
    highest bitrate $\leftarrow \max_{\text{bitrate} \in \text{ladder}}$ bitrate
    pace rate $\leftarrow$ multiplier $\cdot$ highest bitrate
  end if
  return bitrate, pace rate

5 PRODUCTION EVALUATION

We implemented and evaluated Sammy on TV devices (TVs, set-top boxes, game consoles, etc...) in production at Netflix. To evaluate its performance, we ran a series of A/B tests [62, 72] to tune our algorithm and understand the tradeoffs between video QoE and congestion-related metrics. We compared Sammy to Netflix's existing extensively tested and finely-tuned production algorithm, to emphasize how Sammy can improve smoothness while maintaining or improving QoE.

Each A/B test consisted of a control group running Netflix's production algorithm, and twenty treatment groups with different settings of Sammy's parameters. We randomly picked a small fraction of Netflix's users and randomly assigned them to either control or one of the treatment groups. This resulted in a small fraction of Netflix's video sessions (< 1%). We ran the tests for about a week, and measured the values of video- and transport-level metrics for each session. Here we present the results of three experiments. Over all the sessions included in these experiment, Netflix's users watched on the order of thousands of *years* of video.

The experimental results show that Sammy significantly improves smoothness and reduces congestion-related metrics while *slightly improving* video QoE.

Parameter values: We will present results for a single set of parameters for Sammy throughout the rest of the paper, and we briefly discuss other parameter values in Section 5.3. Specifically, Sammy paces at 3.2x the maximum bitrate when the buffer is empty, and 2.8x the maximum bitrate when the buffer is full using the algorithm described in Section 4.2.

As discussed in Section 4.1, Sammy also includes changes to initial throughput estimation. We present the results of these initial changes (without pacing) in Section 5.4.

5.1 Sammy reduces congestion

We first show that Sammy significantly improves smoothness, retransmissions, and round-trip times of Netflix traffic compared to the existing production algorithm. Table 2 presents the percent changes between Sammy and Netflix's production algorithm with 95% confidence intervals.

Type	Metric	% Chg.	95% CI
Congestion	Chunk Throughput	-61.0	[-61.8, -60.2]
	% Retransmits	-35.5	[-37.8, -33.4]
	RTT	-13.7	[-16.4, -12.3]
QoE	Initial VMAF	0.14	[0.1, 0.2]
	VMAF	0.04	[0.0, 0.1]
	Play Delay	-1.29	[-2.0, -0.6]
	Rebuffers (% sess)	–	[-7.1, 4.0]
	Rebuffers (/ hr)	–	[-17.1, 3.6]

Table 2: A/B Test results for Sammy including the percentage change to control and confidence intervals. All statistically significant metric movements are improvements over Netflix's production algorithm.

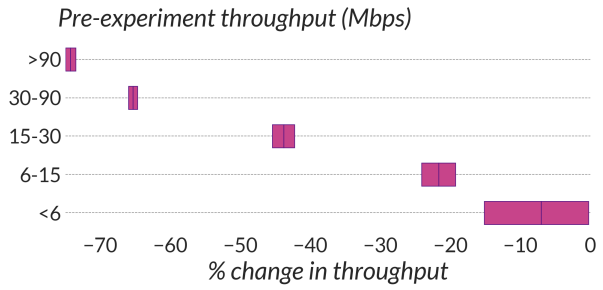


Figure 3: Reduction in chunk throughput (95% CI) split by each user's pre-experiment chunk throughput. Sammy reduces burstiness for users with pre-experiment throughput > 6 Mbps.

Improving smoothness: To measure how much Sammy smooths video traffic, we focus on the average *chunk throughput* (the throughput during "on" periods). Video clients report the average throughput for all chunk downloads in a session, weighted by download time as in Appendix A. We calculate the median of these per-session average chunk throughputs for both Netflix's production algorithm and Sammy. Sammy reduces chunk throughput by 61%. Sammy does not reduce quality, so reducing chunk throughput causes on periods to become longer, and increases the available bandwidth for neighboring traffic during on periods.

Sammy's ability to reduce throughput depends on how much higher network bandwidth is relative to maximum bitrates. This raises the question about how Sammy performs in slower networks. For all users in the A/B test, we computed their pre-experiment throughput by looking at the 95th percentile of their chunk throughput for the week before the test began. We grouped users by the range of pre-experiment throughput: <6 Mbps, 6-15 Mbps, 15-30 Mbps, 30-90 Mbps, and > 90 Mbps. We calculated average chunk throughput within each group of users, and compared Sammy's throughput of each group to that of the production algorithm. Figure 3 shows the percent change in throughput as a function of pre-experiment throughput. For users with pre-experiment throughput of more than 90 Mbps, Sammy reduces chunk throughput by 74%.

As pre-experiment throughput decreases, Sammy reduces chunk throughput less. But Sammy does significantly reduce throughput (improving smoothness) for all pre-experiment throughputs more than 6 Mbps.

Reducing network congestion: Intuitively, reducing chunk throughput and improving smoothness should translate into improvements in congestion-related metrics, specifically lower packet retransmission rates and round-trip times (RTTs). This is supported by our A/B test results.

We calculate the fraction of retransmitted bytes over all bytes sent by TCP for each session. Sammy improves the median fraction of retransmitted bytes over all sessions by 36%. We measure RTTs for each packet sent by TCP and store them for each TCP connection in a t-digest [21]. We merge the t-digests for all TCP connections in a session, and estimate the median RTT for the session. We measure the median of median RTTs over all sessions. Sammy improves RTTs by 14%.

Given Sammy reduces chunk throughput, improves smoothness, and reduces congestion for Netflix traffic, it is plausible that neighboring traffic sharing a bottleneck link with Netflix's traffic should see improvements as well. Section 6 shows in a lab setting that Sammy's improvements in congestion-related metrics can translate to QoE improvements for neighboring traffic.

5.2 Sammy improves QoE

It is not surprising that picking lower pace rates would improve smoothness and network congestion, but more surprisingly, we show this can be done at no cost to the video user experience. In our experiments, Sammy *slightly improves* QoE. These results are summarized in Table 2.

Improving quality and play delay: We measure video quality by Video Multi-method Assessment Fusion (VMAF) [10], a method for estimating a viewer's perception of a video's visual quality. We calculate a time-weighted average of VMAF to get a score for each session, and measure the median score over all sessions. Sammy *slightly* increases overall VMAF, which is driven primarily by an increase in initial VMAF (the VMAF during the first twenty seconds of video playback). In other words, Sammy's video quality is slightly higher than with Netflix's production algorithm.

We note that this is a very minor improvement to VMAF. It is a statistically significant improvement in our experiments, but it is a small and potentially imperceptible improvement. The more important point is that Sammy maintains a QoE that is at least on par to Netflix's existing, finely-tuned production algorithm, with more than 60% lower per-chunk throughput.

Sammy also slightly improves play delay by about 1.3%. This may seem surprising since we do not do any pacing in the initial part of the section. As we show in Section 5.4, the improvements to QoE (play delay included) come primarily from using *only* estimates of initial throughput during the initial phase.

Maintaining rebuffers: Sammy has no statistically significant impact on any other aspect of QoE. There is no significant change in rebuffers: both the fraction of sessions that have at least one rebuffer, and the number of rebuffers per hour streamed.

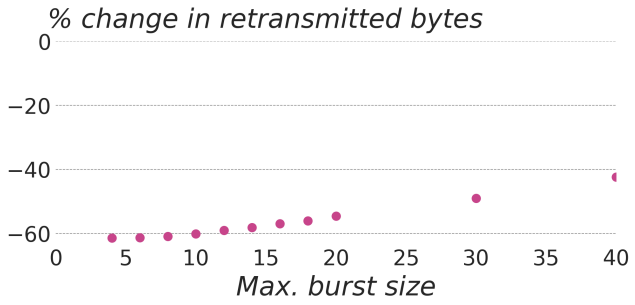


Figure 4: Change in retransmissions as a function of the pacing burst size in a production A/B test. Lower burst sizes improve retransmissions.

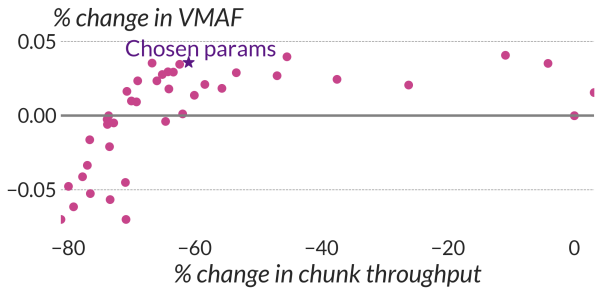


Figure 5: Tradeoff between video quality (VMAF) and chunk throughput for different choices of parameters.

5.3 Tradeoffs and parameter settings

Sammy has a number of parameters that can be tuned, including parameters for pace rate selection and for the chosen ABR algorithm. We used Ax [51] to search the parameter space and find a Pareto improvement to all metrics of interest across multiple rounds of A/B testing.

Different parameter settings allow us to trade off between chunk throughput and quality, as shown in Figure 5. Each point represents one treatment group in an A/B test, each with a different value of parameter settings. The x-axis is the % change in chunk throughput for that group, and the y-axis is the % change in VMAF. The parameters we selected reduced chunk throughput by 61% relative to control, while increasing VMAF by 0.04%. Other parameter settings give other points on this tradeoff. Eventually, decreasing throughput results in a decrease in VMAF.

5.4 QoE differences are primarily from initial phase

Sammy includes two sets of changes over Netflix's production ABR algorithm: reducing chunk throughput using pacing, and changes to the initial phase including using estimates of initial throughput and retuning Netflix's initial bitrate selection logic. Here we report on the results of an A/B test including only the changes to the initial phase, and not pacing.

Metric	% Chg.	95% CI
Initial VMAF	0.30	[0.28,0.35]
VMAF	–	[-1.8e-5, 1.7e-4]
Play Delay	-0.40	[-0.7, -0.1]
Rebuffers (% sess)	–	[-1.6,1.8]
Rebuffers (/ hr)	–	[-3.1,4.2]

Table 3: A/B Test results for Sammy's changes to initial throughput estimation and bitrate selection. All statistically significant metric movements are improvements over Netflix's production algorithm.

In this A/B test, there was a slight improvement in initial VMAF of about 0.3%, in play delay of about 0.4%, and in no other metrics. These results are run as a separate A/B test, and so shouldn't be directly compared to the results of Sammy in Table 2. But the direction of improvement and magnitude is similar in the two tests.

These results, plus our intuition that pacing late in the session should not impact the initial phase (including initial VMAF and play delay), suggests that Sammy's QoE improvements do not come from pacing. Instead, the results suggest that the improvements come primarily from the changes to initial bitrate selection.

5.5 A baseline approach reduces QoE

Sammy works hard to avoid reducing QoE with pacing, and a natural question is whether this work is necessary. Why not just pick a pace rate a bit higher than the maximum bitrate and call it a day? We ran an experiment which shows that this approach underperforms Sammy in all of our goals.

We ran an experiment with the production Netflix ABR algorithm in which we limited the pacing rate for each chunk (including in the initial phase) to 4x the maximum bitrate. We made no other changes. Pacing in this way reduced chunk throughput by 53% and we observed a degradation in most of the major components of video QoE: play delay increased by 6%, and VMAF decreased by 0.2%. The play delay increase was enough to reduce the overall level of streaming, causing the experiment to be automatically stopped by safety systems.

Sammy outperforms this approach in both congestion and QoE-related metrics. Sammy achieves a *higher* reduction in chunk throughput of 61% while *improving* QoE. If we instead chose parameters from Figure 5 which reduced QoE, Sammy would achieve a higher throughput reduction of more than 80% for a much lower VMAF reduction of 0.07%.

5.6 Effect of burst size

With pacing, there is an option of how large a burst to send at a time. To pace at 12 Mbps with 1500 byte MTU, we could send one packet every 1 ms, two packets every 2 ms, or 10 packets every 10ms, and so on. Intuitively, there is a tradeoff in picking the burst size: smaller bursts should improve congestion-related metrics, but also reduce opportunities for segmentation offload which can increase CPU usage.

Netflix's TCP implementation's default behaviour is to limit line-rate bursts to no more than 40 packets at a time. We ran an experiment where we paced at a constant 2x the maximum bitrate, and adjusted the per-packet bursts from 4 packets up to 40 packets. Figure 4 shows the results of this experiment.

Pacing with a burst size of 40 packets corresponds to only reducing chunk throughput, and not reducing the maximum possible size of per-packet bursts. This reduces retransmissions by 40% relative to not pacing. As the maximum burst size decreases, retransmits reduce by up to 60% relative to not pacing. But as the burst size decreases, there is no statistically difference in either chunk throughput or video QoE metrics.

This result shows why it is beneficial to use TCP Pacing instead of capping the congestion window as in prior work [25]. In our experiments, we use a burst size of 4 packets for CPU efficiency. By reducing the burst size from 40 (as it would be if we capped the congestion window) to a burst size of 4, we improve retransmissions by an additional 20%.

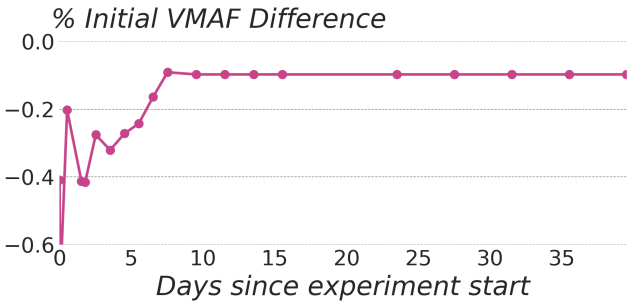


Figure 6: Initial quality difference over time during an A/B test. The treatment algorithm is missing historical data at the beginning of the experiment, and so performs worse over the entire experiment.

5.7 Effect of historical data

As described in Section 4.1, Sammy and prior work use historical throughput measurements for initial bitrate selection. Doing so creates a dependency between successive sessions: the throughput at the beginning of one session impacts the bitrate selection decisions at the beginning of the next. Using historical data improves performance, but the dependency creates challenges for evaluation.

As an example, we ran an experiment simulating introducing a new historical estimate. The treatment group started with no historical measurements, while the control group had historical measurements. Both groups updated historical throughput with the same estimates, and there were no other differences between the groups. Figure 6 shows the percent difference in initial quality over the course of the experiment. The treatment group started with much lower initial quality and surprisingly it stayed lower over the course of the experiment. It took a week for the initial quality of the treatment group to reach its closest point to the control group.

To deal with this challenge, we reset historical throughput information in both treatment and control groups in all experiments to enable an "apples-to-apples" comparison between the two.

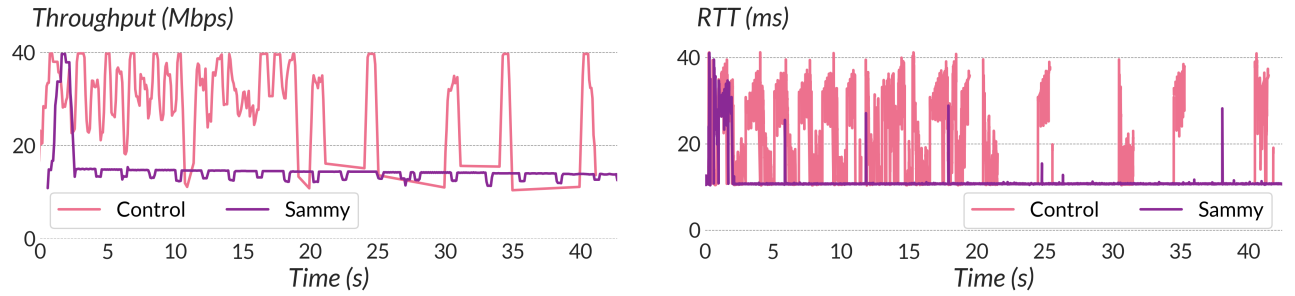
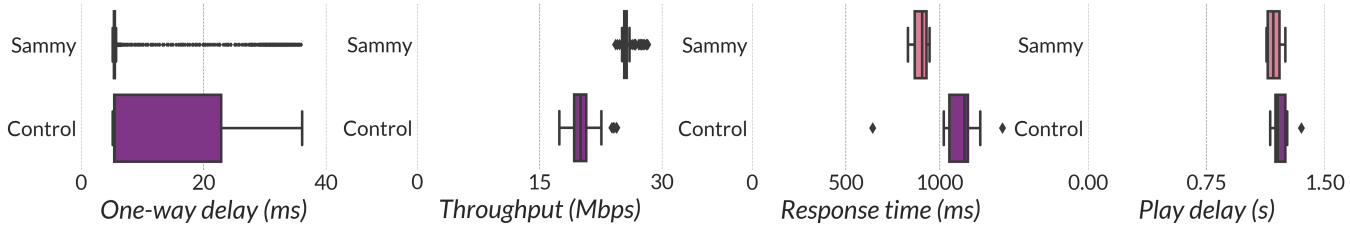


Figure 7: Throughput and RTT for a single Sammy flow running in a lab environment compared to Netflix's production algorithm. After 3 seconds, Sammy reduces chunk throughput enough to avoid congesting the link. Reducing chunk throughput helps it avoid on-off periods beginning for control traffic around 25 seconds.



(a) Neighboring UDP traffic **(b) Neighboring TCP traffic** **(c) Neighboring HTTP traffic** **(d) Neighboring video traffic**

Figure 8: In the lab, Sammy improves QoE for neighboring traffic relative to control for (a) UDP one-way delay, (b) TCP throughput, (c) HTTP response time, and (d) video play delay.

6 IMPROVING QOE OF NEIGHBORS

In this section, we will present lab experiments where we measure how Sammy improves the QoE of neighboring traffic. The previous section shows how Sammy reduces chunk throughput and congestion-related metrics at scale. Lab experiments with a single setting are clearly not representative of most traffic on the internet, so the goal of these experiments is to illustrate how Sammy can improve the QoE of a few neighbors that might share its bottleneck.

Without Sammy, the video traffic fully utilizes the link and fills up the queue, in turn impacting neighboring traffic. Sammy smooths out traffic, and chunk throughput drops to below half the network capacity. This behavior avoids congesting the link (reducing queueing delay for neighboring traffic), and gives neighboring traffic more bandwidth to use for itself.

Experiment setup. In all experiments, we use a 40 Mbps link with a 5ms RTT, and a queue size of 4 times the bandwidth-delay product. Sammy plays a video with a maximum bitrate of 3.3 Mbps. We run an experiment where a video session using Netflix's production algorithm runs at the same time as a neighboring application. We repeat the same experiment using Sammy and observe how the neighbor's QoE changes.

Sammy on its own. To understand how Sammy improves performance for neighboring traffic, we will first look at how it performs on its own. Figure 7 shows the throughput and RTT for a single Sammy flow, compared to a single control flow running Netflix's production ABR algorithm.

At the beginning of the session, both Sammy and control send as fast as possible during the initial phase: fully utilizing the network and filling up the queue. Playback starts after about three seconds, at which point Sammy begins pacing. The pace rate it picks is about 15 Mbps—low enough to avoid congesting the link, so queuing delay goes to zero and the RTT is the minimum of 5ms. Over the rest of the session, Sammy decreases the throughput to about 13 Mbps—below its TCP-fair share rate of 20 Mbps when it shares a link with neighboring traffic.

The change in metrics for this session is comparable to the overall change in metrics for the A/B test in Section 5. For this session Sammy reduces throughput by 53% (slightly less than in the A/B test) and RTTs by 47% (about four times more than the A/B test). Note that in the A/B test, Sammy shares networks with neighboring traffic running typical congestion control algorithms that keep queues full. It is possible that if the neighboring traffic instead used Sammy, the congestion reduction could be even larger [62].

When neighboring traffic shares this particular network with Sammy, it will experience an extra 5 Mbps of available bandwidth and no additional queueing delay. This leads to the following benefits (shown in Figure 8):

UDP: We first run an experiment where the neighboring traffic is a 5 Mbps paced UDP flow. The one-way delay measured for UDP packets is shown in Figure 8a. Sammy eliminates queueing delay for the UDP traffic, reducing the one-way delay by 51%. Without Sammy, video traffic keeps the queue full (see Figure 7) and the UDP traffic experiences queueing delay. Sammy sends no faster

than 15 Mbps during playback, so the queue stays empty even with 5 Mbps of UDP traffic.

TCP: We next run an experiment where the neighboring traffic is a standard, congestion window limited TCP Reno connection (the congestion control algorithm Netflix uses by default). The TCP connection begins 10 seconds after playback starts. Its throughput is shown in Figure 8b. Without Sammy, the TCP flow gets an average of 20 Mbps (its TCP-fair share of throughput). Sammy increases throughput for the TCP flow by 28% to an average of 25.7 Mbps. Any other congestion control algorithms that splits bandwidth equally without Sammy and fully utilizes the link with Sammy would have similar results.

HTTP: The next experiment demonstrates the benefits of Sammy to neighboring HTTP requests. We repeatedly issue 3MB HTTP requests during video playback. We measure the HTTP response time, the time between when the first byte of the request was issued and the last byte of the response was received. The results are shown in Figure 8c—Sammy improves average HTTP response times by 18%, reducing them from 1095ms to 898ms.

Streaming video: We run another experiment to measure the impact of Sammy on a neighboring video session. We start one Sammy session, and after a few seconds we start a neighboring session using Netflix's production algorithm. Figure 8d shows the play delay for the neighboring session. Over four trials, Sammy consistently improves the play delay of its neighbor by 4%—an average of 50ms. Whenever a streaming service shares a bottleneck with itself, Sammy can improve the service's own play delay. This result gives streaming services an incentive to deploy Sammy.

7 CONCLUSION

Our approach shows that ABR algorithms can dramatically reduce the burstiness of video traffic without reducing QoE. In our experiments run at scale at Netflix, Sammy is able to reduce the median chunk throughput by 61%, reducing retransmissions by 36% and RTTs by 14%. These improvements to network congestion came with no harm to video QoE. In fact, we observed a small improvement in video quality (both initially and overall) and play delay, and no statistically significant changes to rebuffers. Because Sammy does not aim to fully utilize the link, there is more bandwidth available for neighboring traffic during Sammy's on periods. Our lab experiments illustrate how this can improve performance for neighboring traffic: Sammy reduces delay for a neighboring UDP flow by 51%, increases throughput for a neighboring TCP flow by 28%, reduces response times for neighboring HTTP traffic by 18%, and even reduces play delay for neighboring video traffic by 4%. We leave deeper investigations of the impact on neighboring traffic to future work, and would be especially interested in experiments to measure the impact at scale.

In many ways, today's video streaming architecture is a *response to* the two control loops managed by ABR and congestion control. Congestion control algorithms learn and acquire their fair share of bandwidth on a packet-by-packet timescale; and in turn ABR algorithms adapt bitrates at chunk-by-chunk timescale. Given the steps taken in this paper, a compelling future path forward is to consider a *single control loop* to both determine the video bitrate and when to transmit each bit of the stream over the network. This algorithm could jointly optimize video QoE and transport-layer goals like congestion and fairness, and could avoid the pitfalls associated with two interacting control loops [30]. We leave that work for others. Here, we instead have the ABR algorithm limit the server's sending rate, so as to allow more rapid deployment with current video streaming services, and keeping with standard practice of sharing the internet using congestion control algorithms. One could also imagine a range of options between the two, where ABR algorithms share more and more information with the underlying transport layer. Broadly, the significant empirical results found in this paper suggest that such innovations have the potential for significant impact not only on video streaming services, but the internet at large.

We view our work as a starting point for using application-level logic to smooth out internet traffic. We have shown that video streaming does not always need the maximum throughput a network can achieve. The layering architecture of the internet encourages other applications to use a similar strategy of allowing congestion control algorithms to select the maximum throughput without application input. By using details about the behavior of other applications, we may be able to make other types of internet traffic into friendlier neighbors as well.

REFERENCES

[1] Amit Aggarwal, Stefan Savage, and Thomas E Anderson. 2000. Understanding the Performance of TCP Pacing. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, Vol. 3. IEEE, Tel Aviv, Israel, 1157–1165. https://doi.org/10.1109/INFCOM.2000.832483

[2] Akamai. 2023. Common Media Client Data & AMD. (Feb. 2023). https://techdocs.akamai.com/adaptive-media-delivery/docs/common-media-client-data-amd

[3] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C. Begen. 2013. Server-Based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video - NOSSDAV '13*. ACM Press, Oslo, Norway, 19–24. https://doi.org/10.1145/2460782.2460786

[4] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: Auto-Tuning Video ABR Algorithms to Network Conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, Budapest Hungary, 44–58. https://doi.org/10.1145/3230543.3230558

[5] João Taveira Araújo, Raúl Landa, Richard G. Clegg, George Pavlou, and Kensuke Fukuda. 2014. A Longitudinal Analysis of Internet Rate Limitations. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 1438–1446. https://doi.org/10.1109/INFCOM.2014.6848078

[6] Consumer Technology Association. 2020. *CTA Specification: Web Application Video Ecosystem - Common Media Client Data*. Technical Report CTA-5004. Consumer Technology Association. https://cdn.cta.tech/cta/media/media/resources/standards/pdfs/cta-5004-final.pdf

[7] Rukshani Athapathu, Ranysha Ware, Aditya Abraham Philip, Srinivasan Seshan, and Justine Sherry. 2020. Prudentia: Measuring Congestion Control Harm on the Internet. In *N2Women Workshop*. 2. http://www.justinesherry.com/papers/athapathu-n2women20.pdf

[8] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. 1998. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98*, Vol. 1. 252–262 vol.1. https://doi.org/10.1109/INFCOM.1998.659661

[9] Abdelhak Bentaleb, May Lim, Mehmet N. Akcay, Ali C. Begen, and Roger Zimmermann. 2021. Common Media Client Data (CMCD): Initial Findings. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, Istanbul Turkey, 25–33. https://doi.org/10.1145/3458306.3461444

[10] Netflix Technology Blog. 2017. Toward A Practical Perceptual Video Quality Metric. (April 2017). https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652

[11] Bob Briscoe. 2007. Flow Rate Fairness: Dismantling a Religion. *ACM SIGCOMM Computer Communication Review* 37, 2 (March 2007), 63–74. https://doi.org/10.

1145/1232919.1232926

[12] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. 2019. When to Use and When Not to Use BBR: An Empirical Analysis and Evaluation Study. In *Proceedings of the Internet Measurement Conference*. ACM, Amsterdam Netherlands, 130–136. https://doi.org/10.1145/3355369.3355579

[13] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-Based Congestion Control. *Commun. ACM* 60, 2 (Jan. 2017), 58–66. https://doi.org/10.1145/3009824

[14] David D Clark. 1988. The Design Philosophy of the DARPA Internet Protocols. *ACM SIGCOMM Computer Communication Review* 18, 4 (Aug. 1988), 106–114. https://doi.org/10.1145/52325.52336

[15] Federal Communications Commission. 2010. *Broadband Performance*. Technical Report OBI Technical Paper No. 4. Federal Communications Commission. https://transition.fcc.gov/national-broadband-plan/broadband-performance-paper.pdf

[16] Federal Communications Commission. 2021. *Measuring Fixed Broadband - Eleventh Report*. Technical Report. Federal Communications Commission. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-eleventh-report

[17] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the Impact of Video Quality on User Engagement. *ACM SIGCOMM Computer Communication Review* 41, 4 (Aug. 2011), 362–373. https://doi.org/10.1145/2043164.2018478

[18] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, P Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *NSDI*. 15.

[19] Eric Dumazet. 2013. Pkt_sched: Fq: Fair Queue Packet Scheduler [LWN.Net]. (Aug. 2013). https://lwn.net/Articles/564825/

[20] Eric Dumazet. 2015. Tc-Fq(8) - Linux Manual Page. (Sept. 2015). https://man7.org/linux/man-pages/man8/tc-fq.8.html

[21] Ted Dunning. 2021. The T-Digest: Efficient Estimates of Distributions. *Software Impacts* 7 (Feb. 2021), 100049. https://doi.org/10.1016/j.simpa.2020.100049

[22] Fastly. 2023. Fastly Developer Hub. (Feb. 2023). https://developer.fastly.com/reference/vcl/variables/client-connection/client-socket-pace/

[23] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video Through Tighter Integration Between a Video Codec and a Transport Protocol. In *NSDI*.

[24] Jim Gettys, Kathleen Nichols, and Kathleen Nichols. 2012. Bufferbloat: Dark Buffers in the Internet. *Commun. ACM* 55, 1 (2012), 15. https://doi.org/10.1145/2063176.2063196

[25] Monia Ghobadi, Yuchung Cheng, Ankur Jain, and Matt Mathis. 2012. Trickle: Rate Limiting YouTube Video Streaming. In *USENIX ATC*. 6.

[26] Carlo Augusto Grazia, Martin Klapez, and Maurizio Casoni. 2021. The New TCP Modules on the Block: A Performance Evaluation of TCP Pacing and TCP Small Queues. *IEEE Access* 9 (2021), 129329–129336. https://doi.org/10.1109/ACCESS.2021.3113891

[27] Bruce Hajek. 1983. The Proof of a Folk Theorem on Queuing Delay with Applications to Routing in Networks. *J. ACM* 30, 4 (Oct. 1983), 834–851. https://doi.org/10.1145/2157.322409

[28] Janey C. (Janey Ching-Iu) Hoe. 1995. *Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes*. Thesis. Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/36971

[29] Stefan Holmer, Henrik Lundin, Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. 2015. *A Google Congestion Control Algorithm for Real-Time Communication*. Internet Draft draft-alvestrand-rmcat-congestion-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-alvestrand-rmcat-congestion-03

[30] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, Timid, and Unstable: Picking a Video Streaming Rate Is Hard. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference - IMC '12*. ACM Press, Boston, Massachusetts, USA, 225. https://doi.org/10.1145/2398776.2398800

[31] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM - SIGCOMM '14*. ACM Press, Chicago, Illinois, USA, 187–198. https://doi.org/10.1145/2619239.2626296

[32] Per Hurtig, Habtegebreil Haile, Karl-Johan Grinnemo, Anna Brunstrom, Eneko Atxutegi, Fidel Liberal, and Ake Arvidsson. 2018. Impact of TCP BBR on CUBIC Traffic: A Mixed Workload Evaluation. In *2018 30th International Teletraffic Congress (ITC 30)*. IEEE, Vienna, 218–226. https://doi.org/10.1109/ITC30.2018.00040

[33] Geoff Huston. 2018. TCP and BBR. (May 2018). https://ripe76.ripe.net/presentations/10-2018-05-15-bbr.pdf

[34] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. 2016. CFA: A Practical Prediction System for Video QoE Optimization. In *NSDI*.

[35] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Transactions on Networking* 22, 1 (Feb. 2014), 326–340. https://doi.org/10.1109/TNET.2013.2291681

[36] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. 2017. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *Proceedings of the 2017 Internet Measurement Conference*. ACM, London United Kingdom, 290–303. https://doi.org/10.1145/3131365.3131368

[37] Seong-ryong Kang, Xiliang Liu, Min Dai, and Dmitri Loguinov. 2004. Packet-Pair Bandwidth Estimation - Stochastic Analysis of a Single Congested Node. *ICNP* (2004), 316–325. https://doi.org/10.1109/ICNP.2004.1348121

[38] Damian Karwowski, Tomasz Grajek, Krzysztof Klimaszewski, Olgierd Stankiewicz, Jakub Stankowski, and Krzysztof Wegner. 2017. 20 Years of Progress in Video Compression – from MPEG-1 to MPEG-H HEVC. General View on the Path of Video Coding Development. In *International Conference on Image Processing and Communications*, Vol. 525. 3–15. https://doi.org/10.1007/978-3-319-47274-4_1

[39] S Keshav. 1995. A Control-Theoretic Approach to Flow Control. *ACM SIGCOMM Computer Communication Review* (1995). http://dl.acm.org/citation.cfm?id=205463

[40] S. Shunmuga Krishnan and Ramesh K. Sitaraman. 2012. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. In *Proceedings of the 2012 Internet Measurement Conference (IMC '12)*. Association for Computing Machinery, New York, NY, USA, 211–224. https://doi.org/10.1145/2398776.2398799

[41] Jonathan Kua, Grenville Armitage, and Philip Branch. 2017. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1842–1866. https://doi.org/10.1109/COMST.2017.2685630

[42] Ike Kunze, Jan Ruth, and Oliver Hohlfeld. 2020. Congestion Control in the Wild—Investigating Content Provider Fairness. *IEEE Transactions on Network and Service Management* 17, 2 (June 2020), 1224–1238. https://doi.org/10.1109/TNSM.2019.2962607

[43] Will Law. 2022. Clever Monkeys Communicating Discreetly. (Oct. 2022). https://2022.demuxed.com/

[44] Ahmed Mansy, Bill Ver Steeg, and Mostafa Ammar. 2013. SABRE: A Client Based Technique for Mitigating the Buffer Bloat Effect of Adaptive Video Flows. In *Proceedings of the 4th ACM Multimedia Systems Conference on - MMSys '13*. ACM Press, Oslo, Norway, 214–225. https://doi.org/10.1145/2483977.2484004

[45] Aditya Mavlankar, Liwei Guo, Anush Moorthy, and Anne Aaron. 2020. Optimized Shot-Based Encodes for 4K: Now Streaming! (Aug. 2020). https://netflixtechblog.com/optimized-shot-based-encodes-for-4k-now-streaming-47b516b10bbb

[46] Tong Meng, Neta Rozen Schiff, P. Brighten Godfrey, and Michael Schapira. 2020. PCC Proteus: Scavenger Transport And Beyond. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 615–631. https://doi.org/10.1145/3387514.3405891

[47] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, London United Kingdom, 537–550. https://doi.org/10.1145/2785956.2787510

[48] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. 2019. End-to-End Transport for Video QoE Fairness. In *Proceedings of the ACM Special Interest Group on Data Communication - SIGCOMM '19*. ACM Press, Beijing, China, 408–423. https://doi.org/10.1145/3341302.3342077

[49] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. 2021. VOXEL: Cross-Layer Optimization for Video Streaming with Imperfect Transmission. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*. ACM, Virtual Event Germany, 359–374. https://doi.org/10.1145/3485983.3494864

[50] Larry Peterson, Lawrence Brakmo, and Bruce Davie. 2022. *TCP Congestion Control: A Systems Approach* (version 1.1-dev ed.). Self-published. https://tcpcc.systemsapproach.org/

[51] Meta Platforms. 2023. Ax: Adaptive Experimentation Platform. (Feb. 2023). https://ax.dev

[52] Michele Polese, Federico Chiariotti, Elia Bonetto, Filippo Rigotto, Andrea Zanella, and Michele Zorzi. 2019. A Survey on Recent Advances in Transport Layer Protocols. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3584–3608. https://doi.org/10.1109/COMST.2019.2932905 arXiv:cs/1810.03884

[53] World Snail Racing. 2023. World Snail Racing Championships. (Feb. 2023). http://www.snailracing.net/

[54] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. 2011. Network Characteristics of Video Streaming Traffic. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*. ACM, Tokyo Japan, 1–12. https://doi.org/10.1145/2079296.2079321

[55] Dario Rossi, Claudio Testa, Silvio Valenti, and Luca Muscariello. 2016. LEDBAT: The New BitTorrent Congestion Control Protocol. In *2010 19th International Conference on Computer Communications and Networks (ICCCN 2010)*. IEEE, 1–6. https://doi.org/10.1109/ICCCN.2010.5560080

[56] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable Traffic Shaping at End Hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, Los Angeles CA USA, 404–417. https://doi.org/10.1145/3098822.3098852

[57] Sandvine. 2023. *Sandivne Global Internet Phenomena Report 2023*. Technical Report. Sandvine.

[58] Yusuf Sani, Andreas Mauthe, and Christopher Edwards. 2017. Adaptive Bitrate Selection: A Survey. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2985–3014. https://doi.org/10.1109/COMST.2017.2725241

[59] Kozo Satoda, Hiroshi Yoshida, Hironori Ito, and Kazunori Ozawa. 2012. Adaptive Video Pacing Method Based on the Prediction of Stochastic TCP Throughput. In *2012 IEEE Global Communications Conference (GLOBECOM)*. 1944–1950. https://doi.org/10.1109/GLOCOM.2012.6503400

[60] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. 2018. Towards a Deeper Understanding of TCP BBR Congestion Control. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, Zurich, Switzerland, 1–9. https://doi.org/10.23919/IFIPNetworking.2018.8696830

[61] Bruce Spang. 2023. Sammy: Making Video Traffic a Friendlier Internet Neighbor. (June 2023). https://sammy.brucespang.com

[62] Bruce Spang, Veronica Hannan, Shravya Kunamalla, Te-Yuan Huang, Nick McKeown, and Ramesh Johari. 2021. Unbiased Experiments in Congested Networks. In *Proceedings of the 21st ACM Internet Measurement Conference (IMC '21)*. Association for Computing Machinery, New York, NY, USA, 80–95. https://doi.org/10.1145/3487552.3487851

[63] Speedtest. 2023. Internet Speed around the World. (Feb. 2023). https://www.speedtest.net/global-index

[64] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2019. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. *ACM Transactions on Multimedia Computing, Communications, and Applications* 15, 2s (April 2019), 1–29. https://doi.org/10.1145/3336497

[65] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking* 28, 4 (Aug. 2020), 1698–1711. https://doi.org/10.1109/TNET.2020.2996964

[66] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, Florianopolis Brazil, 272–285. https://doi.org/10.1145/2934872.2934898

[67] Linus Torvalds. 2021. Tcp_input.c - Linux (v5.11-Rc5). (Jan. 2021). https://github.com/torvalds/linux/blob/2ab38c17aac10bf55ab3efde4c4db3893d8691d2/net/ipv4/tcp_input.c#L873

[68] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. 2019. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. *arXiv:1903.03852 [cs]* (March 2019). arXiv:cs/1903.03852 http://arxiv.org/abs/1903.03852

[69] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. 2019. Interactions between Congestion Control Algorithms. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*. 161–168. https://doi.org/10.23919/TMA.2019.8784674

[70] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of the Internet Measurement Conference*. ACM, Amsterdam Netherlands, 137–143. https://doi.org/10.1145/3355369.3355604

[71] David X. Wei, Pei Cao, and Steven H. Low. 2006. TCP Pacing Revisited. In *INFOCOM*, Vol. 2. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.2658&rep=rep1&type=pdf

[72] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in Situ: A Randomized Experiment in Video Streaming. In *NSDI*. Santa Clara, CA, USA, 16. https://www.usenix.org/system/files/nsdi20-paper-yan.pdf

[73] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. Association for Computing Machinery, New York, NY, USA, 325–338. https://doi.org/10.1145/2785956.2787486

[74] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. 2017. An Axiomatic Approach to Congestion Control. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, Palo Alto CA USA, 115–121. https://doi.org/10.1145/3152434.3152445

[75] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. 2021. SENSEI: Aligning Video Streaming Quality with Dynamic User Sensitivity. In *NSDI*.

[76] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. 2016. ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*. ACM, Irvine California USA, 313–327. https://doi.org/10.1145/2999572.2999593

A APPENDIX: RELATIONSHIP BETWEEN THROUGHPUT, BITRATE, AND BUFFERS

Appendices are supporting material that has not been peer-reviewed.

In this appendix, we will formalize the relationship between chunk throughput, bitrates, and buffer sizes.

There are a number of chunk selection opportunities, which occur at steps $t \in \{1, \ldots, T\}$. The chunk at time t has a duration d_t and size s_t which is selected by the ABR algorithm.

One step: Each time we select a chunk at time t, the buffer will evolve in some way. Let Δ_t be the time it takes to add chunk t to the buffer. For simplicity we will assume the buffer never becomes full and never becomes empty, but we could instead keep track of the amount of full and empty time after each chunk downloads.

The buffer evolution is given by the following standard equation [30]:

$$B_{t+1} = B_t + d_t - \Delta_t. \tag{2}$$

We will define the bitrate of chunk t as

$$r_t = \frac{s_t}{d_t}. \tag{3}$$

We will define x_t, the throughput of chunk t (e.g. in units of bits per second), as

$$x_t = \frac{s_t}{\Delta_t} = \frac{r_t d_t}{\Delta_t}. \tag{4}$$

Note that with these definitions,

$$B_{t+1} = B_t + d_t - d_t \frac{r_t}{x_t}. \tag{5}$$

Multiple steps: In addition to a single buffer step, we will also be interested in how the buffer evolves over T steps. Define the total duration D_T as

$$D_T = \sum_{t=1}^{T} d_t.$$

When playback starts, the buffer starts at some size B_0. If we expand (2), we have the following buffer size at time T+1. We could interpret this as being the buffer right after we finish downloading the last chunk.

$$B_{T+1} = B_0 + D_T - \sum_{t=1}^{T} \Delta_t. \tag{6}$$

We will define S_T to be the total size of chunks we download by time T

$$S_T = \sum_{t=1}^{T} s_t. \tag{7}$$

Define the time-average bitrate by

$$\bar{r} = \frac{\sum_{t=1}^{T} d_t r_t}{D_T} = \frac{S_T}{D_T}. \tag{8}$$

And finally we will define the time-average throughput as:

$$\bar{x} = \frac{\sum_{t=1}^{T} \Delta_t x_t}{\sum_{t=1}^{T} \Delta_t} = \frac{S_T}{\sum_{t=1}^{T} \Delta_t}. \tag{9}$$

With these definitions, the behavior of the buffer over T steps is the same as the behavior over one step, averaged. This is formalized

by the following theorem, which should be compared to the single step update equation in (5).

Theorem A.1. *In the above setting,*

$$B_{T+1} = B_0 + D_T - D_T \frac{\bar{r}}{\bar{x}}.$$

Proof. Given (6), all we need to show is that $D_T \frac{\bar{r}}{\bar{x}} = \sum_{t=1}^{T} \Delta_t$. Substituting the definition of $\bar{r}$, we have

$$D_T \frac{\bar{r}}{\bar{x}} = D_T \frac{S_T/D_T}{\bar{x}} = \frac{S_T}{\bar{x}}.$$

By the definition of $\bar{x}$, we have

$$D_T \frac{\bar{r}}{\bar{x}} = \frac{S_T}{S_T/\sum_{t=1}^{T} \Delta_t} = \sum_{t=1}^{T} \Delta_t.$$

□

A.1 Discussion

The main use of this theorem in our paper is to understand which bitrates our algorithm will pick, by understanding how a simulated buffer evolves as a function of bitrate and throughput. But this theorem is a much more general statement about how the playback buffer evolves. Note that the only critical assumption we have made is that (2) holds. Our definition of bitrates r_t and throughput x_t ensures that (4) follows from (2).

In this section, we will point out some of the consequences of the Theorem for ABR algorithms.

A.1.1 Cannot exceed average throughput without buffer help. Intuition tells us average bitrate cannot exceed average throughput. Theorem A.1 gives us a simple formalization.

Say that the buffer does not decrease, so $B_0 \leq B_{T+1}$. Then

$$1 - \frac{B_{T+1} - B_0}{D_T} \leq 1.$$

By Theorem A.1, $\bar{r} \leq \bar{x}$. That is, the bitrate cannot exceed average throughput.

However if we reduce the size of the buffer, we can exceed average throughput. Suppose $B_0 \geq B_{T+1}$. In this case,

$$1 - \frac{B_{T+1} - B_0}{D_T} \geq 1.$$

By Theorem A.1, $\bar{r} \geq \bar{x}$.

A.1.2 Building up a buffer comes at the expense of bitrate. Suppose we have built up a 5 minute buffer by the time we select the last chunk ($B_0 = 0$, $B_{T+1} = 300$), then rearranging Theorem A.1 gives:

$$\bar{r} = \bar{x}\left(1 - \frac{5}{D_T}\right).$$

Over a twenty minute session, this says that $\bar{r} = 0.75x$. Restating, if an ABR algorithm builds up a 5 minute buffer over a 20 minute session then it will get a bitrate which is 75% of average throughput.

A.1.3 Intermediate buffer values do not affect average bitrate. All the terms in Theorem A.1 are averages, the difference between ending and starting buffer, and the duration. From the perspective of the average bitrate we can achieve, it doesn't matter if the throughput is stable or wildly variable. The path of the buffer is also not important—the only terms that affect bitrate are the starting and ending buffer sizes. We can build up a large intermediate buffer by picking a lower bitrate than throughput, and then decrease the buffer to regain bitrate.

As an example, suppose we start with no buffer and build up a thirty second buffer during the first sixty seconds of playback. By Theorem A.1, over the first sixty seconds $\bar{r} = 0.5\bar{x}$. Suppose we make careful choices over the rest of the session, and keep the buffer at thirty seconds after twenty minutes of playback. By Theorem A.1, $\bar{r} = 0.975\bar{x}$. By controlling the size of the buffer over the course of the session, we don't suffer for our early choice to build up a large buffer. This effect is what allows buffer-based algorithms to achieve high bitrates.

Your Botnet is My Botnet: Analysis of a Botnet Takeover

Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski,
Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna
Department of Computer Science, University of California, Santa Barbara
{bstone,marco,sullivan,rgilbert,msz,kemm,chris,vigna}@cs.ucsb.edu

ABSTRACT

Botnets, networks of malware-infected machines that are controlled by an adversary, are the root cause of a large number of security problems on the Internet. A particularly sophisticated and insidious type of bot is Torpig, a malware program that is designed to harvest sensitive information (such as bank account and credit card data) from its victims. In this paper, we report on our efforts to take control of the Torpig botnet and study its operations for a period of ten days. During this time, we observed more than 180 thousand infections and recorded almost 70 GB of data that the bots collected. While botnets have been "hijacked" and studied previously, the Torpig botnet exhibits certain properties that make the analysis of the data particularly interesting. First, it is possible (with reasonable accuracy) to identify unique bot infections and relate that number to the more than 1.2 million IP addresses that contacted our command and control server. Second, the Torpig botnet is large, targets a variety of applications, and gathers a rich and diverse set of data from the infected victims. This data provides a new understanding of the type and amount of personal information that is stolen by botnets.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software*

General Terms

Security

Keywords

Botnet, Malware, Measurement, Security, Torpig

1. INTRODUCTION

Malicious code (or malware) has become one of the most pressing security problems on the Internet. In particular, this is true for bots [5], a type of malware that is written with the intent of taking over a large number of hosts on the Internet. Once infected with a bot, the victim host will join a botnet, which is a network of compromised machines that are under the control of a malicious entity, typically referred to as the botmaster. Botnets are the primary means for cyber-criminals to carry out their nefarious tasks, such as sending spam mails [36], launching denial-of-service attacks [29], or stealing personal data such as mail accounts or bank credentials [16, 39]. This reflects the shift from an environment in which malware was developed for fun, to the current situation, where malware is spread for financial profit.

Given the importance of the problem, significant research effort has been invested to gain a better understanding of the botnet phenomenon.

One approach to study botnets is to perform *passive analysis* of secondary effects that are caused by the activity of compromised machines. For example, researchers have collected spam mails that were likely sent by bots [47]. Through this, they were able to make indirect observations about the sizes and activities of different spam botnets. Similar measurements focused on DNS queries [34, 35] or DNS blacklist queries [37] performed by bot-infected machines. Other researchers analyzed network traffic (netflow data) at the tier-1 ISP level for cues that are characteristic for certain botnets (such as scanning or long-lived IRC connections) [24]. While the analysis of secondary effects provides interesting insights into particular botnet-related behaviors, one can typically only monitor a small portion of the Internet. Moreover, the detection is limited to those botnets that actually exhibit the activity targeted by the analysis.

A more *active* approach to study botnets is via *infiltration*. That is, using an actual malware sample or a client simulating a bot, researchers join a botnet to perform analysis from the inside. To achieve this, honeypots, honey clients, or spam traps are used to obtain a copy of a malware sample. The sample is then executed in a controlled environment, which makes it possible to observe the traffic that is exchanged between the bot and its command and control (C&C) server(s). In particular, one can record the commands that the bot receives and monitor its malicious activity. For some botnets that rely on a central IRC-based C&C server, joining a botnet can also reveal the IP addresses of other clients (bots) that are concurrently logged into the IRC channel [4, 11, 35]. While this technique worked well for some time, attackers have unfortunately adapted, and most current botnets use stripped-down IRC or HTTP servers as their centralized command and control channels. With such C&C infrastructures, it is no longer possible to make reliable statements about other bots by joining as a client.

Interestingly, due to the open, decentralized nature of peer-to-peer (P2P) protocols, it is possible to infiltrate P2P botnets such as Storm. To this end, researchers have developed crawlers that actively search the P2P network for client nodes that exhibit bot-like characteristics. Such crawls are the basis for studying the num-

CCS'09, November 9–13, 2009, Chicago, Illinois, USA.

ber of infected machines [18, 21] and the ways in which criminals orchestrate spam campaigns [23]. Of course, the presented techniques only work in P2P networks that can be actively crawled. Thus, they are not applicable to a majority of current botnets, which rely mostly on a centralized IRC or HTTP C&C infrastructure.

To overcome the limitations of passive measurements and infiltration – in particular in the case of centralized IRC and HTTP botnets – one can attempt to *hijack* the entire botnet, typically by taking control of the C&C channel. One way to achieve this is to directly seize the physical machines that host the C&C infrastructure [8]. Of course, this is only an option for law enforcement agencies. Alternatively, one can tamper with the domain name service (DNS), as bots typically resolve domain names to connect to their command and control infrastructure. Therefore, by collaborating with domain registrars (or other entities such as dynamic DNS providers), it is possible to change the mapping of a botnet domain to point to a machine controlled by the defender [6]. Finally, several recent botnets, including Torpig, use the concept of *domain flux*. With domain flux, each bot periodically (and independently) generates a list of domains that it contacts. The bot then proceeds to contact them one after another. The first host that sends a reply that identifies it as a valid C&C server is considered genuine, until the next period of domain generation is started. By reverse engineering the domain generation algorithm, it is possible to pre-register domains that bots will contact at some future point, thus denying access to the botmaster and redirecting bot requests to a server under one's own control. This provides a unique view on the entire infected host population and the information that is collected by the botmasters.

In this paper, we describe our experience in actively seizing control of the Torpig (*a.k.a.* Sinowal, or Anserin) botnet for ten days. Torpig, which has been described in [40] as "one of the most advanced pieces of crimeware ever created," is a type of malware that is typically associated with bank account and credit card theft. However, as we will see, it also steals a variety of other personal information.

As mentioned previously, the Torpig botnet makes use of domain flux to locate active C&C servers. To take over this botnet, we leveraged information about the domain generation algorithm and Torpig's C&C protocol to register domains that the infected hosts would contact. By providing a valid response, the bots accepted our server as genuine, and volunteered a wealth of information, which we collected and analyzed. This is an approach that is similar to botnet takeover attempts of the Kraken [1] and Conficker [32] botnets. However, in contrast to previous takeovers, we observe that Torpig has certain properties that make our analysis particularly interesting.

First, Torpig bots transmit identifiers that permit us to distinguish between individual infections. This is different from other botnets such as Conficker. The presence of unique identifiers allows us to perform a precise estimate of the botnet size. Moreover, we can account for DHCP churn and NAT effects, which are well-known problems when computing botnet sizes. In addition, we compare our results to IP-based techniques that are commonly used to estimate botnet populations.

Second, Torpig is a data harvesting bot that targets a wide variety of applications and extracts a wealth of information from the infected victims. Together with the large size of the botnet (we observed more than 180 thousand infections), we have access to a rich data set that sheds light on the quantity and nature of the data that cyber-criminals can harvest, the financial profits that they can make, and the threats to the security and privacy of bot victims. The availability of this rich data set is different from previous work where the authors could not send valid responses to the bots (because C&C messages are authenticated [32]) or where the bots were simply not collecting such information [1].

In summary, the main contribution of this paper is a comprehensive analysis of the operations of the Torpig botnet. For ten days, we obtained information that was sent by more than 180 thousand infected machines. This data provides a vivid demonstration of the threat that botnets in general, and Torpig in particular, present to today's Internet. For our paper, we study the size of the botnet and compare our results to alternative ways of counting botnet populations. In addition, the analysis of the rich and diverse collection of user data provides a new understanding of the type and amount of personal information that is stolen by botnets.

2. BACKGROUND

Torpig is a malware that has drawn much attention recently from the security community. On the surface, it is one of the many Trojan horses infesting today's Internet that, once installed on the victim's machine, steals sensitive information and relays it back to its controllers. However, the sophisticated techniques it uses to steal data from its victims, the complex network infrastructure it relies on, and the vast financial damage that it causes set Torpig apart from other threats.

So far, Torpig has been distributed to its victims as part of Mebroot. Mebroot is a rootkit that takes control of a machine by replacing the system's Master Boot Record (MBR). This allows Mebroot to be executed at boot time, before the operating system is loaded, and to remain undetected by most anti-virus tools. More details on Mebroot can be found in [9, 12, 25]. In this paper, we will focus on Torpig, introducing Mebroot only when necessary to understand Torpig's behavior. In particular, hereinafter, we present the life cycle of Torpig and the organization of the Torpig botnet, as we observed it during the course of our analysis. We will use Figure 1 as a reference.

Victims are infected through drive-by-download attacks [33]. In these attacks, web pages on legitimate but vulnerable web sites (1) are modified with the inclusion of HTML tags that cause the victim's browser to request JavaScript code (2) from a web site (the *drive-by-download server* in the figure) under control of the attackers (3). This JavaScript code launches a number of exploits against the browser or some of its components, such as ActiveX controls and plugins. If any exploit is successful, an executable is downloaded from the drive-by-download server to the victim machine, and it is executed (4).

The downloaded executable acts as an installer for Mebroot. The installer injects a DLL into the file manager process (`explorer.exe`), and execution continues in the file manager's context. This makes all subsequent actions appear as if they were performed by a legitimate system process. The installer then loads a kernel driver that wraps the original disk driver (`disk.sys`). At this point, the installer has raw disk access on the infected machine. The installer can then overwrite the MBR of the machine with Mebroot. After a few minutes, the machine automatically reboots, and Mebroot is loaded from the MBR.

Mebroot has no malicious capability *per se*. Instead, it provides a generic platform that other modules can leverage to perform their malicious actions. In particular, Mebroot provides functionality to manage (install, uninstall, and activate) such additional modules. Immediately after the initial reboot, Mebroot contacts the *Mebroot C&C server* to obtain malicious modules (5). These modules are saved in encrypted form in the `system32` directory, so that, if the user reboots the machine, they can be immediately reused without having to contact the C&C server again. The saved modules

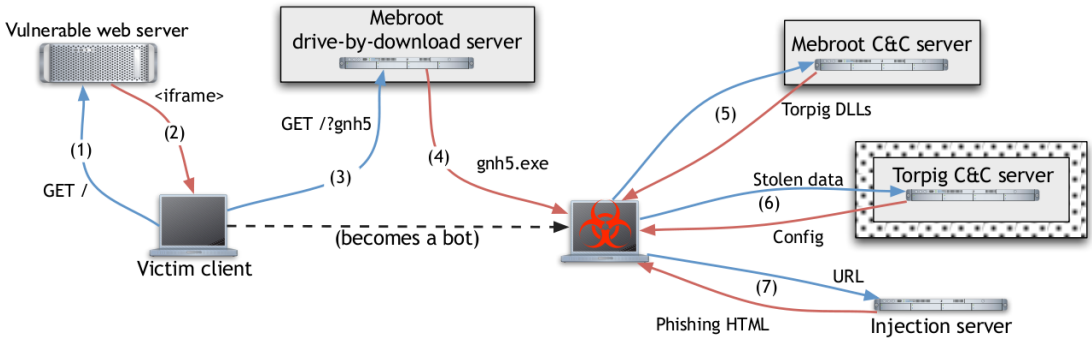


Figure 1: The Torpig network infrastructure. Shaded in gray are the components for which a domain generation algorithm is used. The component that we "hijacked" is shown with dotted background.

are timestamped and named after existing files in the same directory (they are given a different, random extension), to avoid raising suspicion. After the initial update, Mebroot contacts its C&C server periodically, in two-hour intervals, to report its current configuration (i.e., the type and version number of the currently installed modules) and to potentially receive updates. All communication with the C&C server occurs via HTTP requests and responses and is encrypted using a sophisticated, custom encryption algorithm [9]. Currently, no publicly available tool exists to circumvent this encryption scheme.

During our monitoring, the C&C server distributed three modules, which comprise the Torpig malware. Mebroot injects these modules (i.e., DLLs) into a number of applications. These applications include the Service Control Manager (`services.exe`), the file manager, and 29 other popular applications, such as web browsers (e.g., Microsoft Internet Explorer, Firefox, Opera), FTP clients (CuteFTP, LeechFTP), email clients (e.g., Thunderbird, Outlook, Eudora), instant messengers (e.g., Skype, ICQ), and system programs (e.g., the command line interpreter `cmd.exe`). After the injection, Torpig can inspect all the data handled by these programs and identify and store interesting pieces of information, such as credentials for online accounts and stored passwords.

Periodically (every twenty minutes, during the time we monitored the botnet), Torpig contacts the *Torpig C&C server* to upload the data stolen since the previous reporting time (6). This communication with the server is also over HTTP and is protected by a simple obfuscation mechanism, based on XORing the clear text with an 8-byte key and base64 encoding. This scheme was broken by security researchers at the end of 2008, and tools are available to automate the decryption [20]. The C&C server can reply to a bot in one of several ways. The server can simply acknowledge the data. We call this reply an `okn` response, from the string contained in the server's reply. In addition, the C&C server can send a configuration file to the bot (we call this reply an `okc` response). The configuration file is obfuscated using a simple XOR-11 encoding. It specifies how often the bot should contact the C&C server, a set of hard-coded servers to be used as backup, and a set of parameters to perform "man-in-the-browser" phishing attacks [14].

Torpig uses phishing attacks to actively elicit additional, sensitive information from its victims, which, otherwise, may not be observed during the passive monitoring it normally performs. These attacks occur in two steps. First, whenever the infected machine visits one of the domains specified in the configuration file (typically, a banking web site), Torpig issues a request to an *injection server*. The server's response specifies a page on the target domain where the attack should be triggered (we call this page the *trigger page*, and it is typically set to the login page of a site), a URL on the injection server that contains the phishing content (the *injection URL*), and a number of parameters that are used to fine tune the attack (e.g., whether the attack is active and the maximum number of times it can be launched). The second step occurs when the user visits the trigger page. At that time, Torpig requests the injection URL from the injection server and injects the returned content into the user's browser (7). This content typically consists of an HTML form that asks the user for sensitive information, for example, credit card numbers and social security numbers.

These phishing attacks are very difficult to detect, even for attentive users. In fact, the injected content carefully reproduces the style and look-and-feel of the target web site. Furthermore, the injection mechanism defies all phishing indicators included in modern browsers. For example, the SSL configuration appears correct, and so does the URL displayed in the address bar. An example screen-shot of a Torpig phishing page for Wells Fargo Bank is shown in Figure 2. Notice that the URL correctly points to `https://online.wellsfargo.com/signon`, the SSL certificate has been validated, and the address bar displays a padlock. Also, the page has the same style as the original web site.

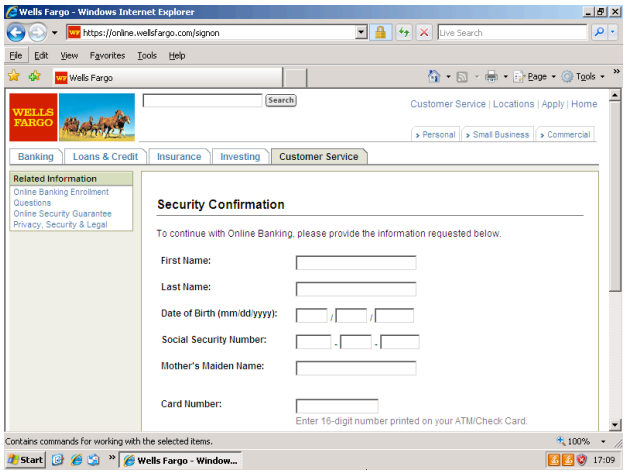


Figure 2: A man-in-the-browser phishing attack.

Communication with the injection server is protected using the standard HTTPS protocol. However, since Torpig does not check the validity of the server's certificate and blindly accepts any self-

signed certificate, it is possible to mount a man-in-the-middle attack and recover the data exchanged with the injection server.

In summary, Torpig relies on a fairly complex network infrastructure to infect machines, retrieve updates, perform active phishing attacks, and send the stolen information to its C&C server. However, we observed that the schemes used to protect the communication in the Torpig botnet (except those used by the Mebroot C&C) are insufficient to guarantee basic security properties (confidentiality, integrity, and authenticity). This was a weakness that enabled us to seize control of the botnet.

3. DOMAIN FLUX

A fundamental aspect of any botnet is that of coordination; i.e., how the bots identify and communicate with their C&C servers. Traditionally, C&C hosts have been located by their bots using their IP address, DNS name, or their node ID in peer-to-peer overlays. In the recent past, botnet authors have identified several ways to make these schemes more flexible and robust against take-down actions, e.g., by using IP fast-flux techniques [17]. With fast-flux, the bots would query a certain domain that is mapped onto a set of IP addresses, which change frequently. This makes it more difficult to take down or block a specific C&C server. However, fast-flux uses only a single domain name, which constitutes a single point of failure.

Torpig solves this issue by using a different technique for locating its C&C servers, which we refer to as *domain flux*. With domain flux, each bot uses a domain generation algorithm (DGA) to compute a list of domain names. This list is computed independently by each bot and is regenerated periodically. Then, the bot attempts to contact the hosts in the domain list in order until one succeeds, i.e., the domain resolves to an IP address and the corresponding server provides a response that is valid in the botnet's protocol. If a domain is blocked (for example, the registrar suspends it to comply with a take-down request), the bot simply rolls over to the following domain in the list. Domain flux is also used to contact the Mebroot C&C servers and the drive-by-download servers. Domain flux is increasingly popular among botnet authors. In fact, similar mechanisms were used before by the Kraken/Bobax [1] and the Srizbi bots [46], and, more recently, by the Conficker worm [32].

In Torpig, the DGA is seeded with the current date and a numerical parameter. The algorithm first computes a "weekly" domain name, say *dw*, which depends on the current week and year, but is independent of the current day (i.e., remains constant for the entire week). Using the generated domain name *dw*, a bot appends a number of TLDs: in order, *dw*.com, *dw*.net, and *dw*.biz. It then resolves each domain and attempts to connect to its C&C server. If all three connections fail, Torpig computes a "daily" domain, say *dd*, which in addition depends on the current day (i.e., a new domain *dd* is generated each day). Again, *dd*.com is tried first, with fallbacks to *dd*.net and *dd*.biz. If these domains also fail, Torpig attempts to contact the domains hardcoded in its configuration file (e.g., `rikora.com`, `pinakola.com`, and `flippibi.com`). Listing 1 shows the pseudo-code of the routines used to generate the daily domains *dd*. The DGA used in Torpig is completely deterministic; i.e., once the current date is determined, all bots generate the same list of domains, in the same order.

From a practical standpoint, domain flux generates a list of "rendezvous points" that *may* be used by the botmasters to control their bots. Not all the domains generated by a DGA need to be valid for the botnet to be operative. However, there are two requirements that the botmasters must satisfy to maintain their grip on the botnet. First, they must control at least one of the domains that will be contacted by the bots. Second, they must use mechanisms to prevent other groups from seizing domains that will be contacted by bots before the domains under their control.

```
suffix =  ["anj", "ebf", "arm", "pra", "aym", "unj",
    "ulj", "uag", "esp", "kot", "onv", "edc"]

def generate_daily_domain():
    t = GetLocalTime()
    p = 8
    return generate_domain(t, p)

def scramble_date(t, p):
    return (((t.month ^ t.day) + t.day) * p) +
        t.day + t.year

def generate_domain(t, p):
    if t.year < 2007:
        t.year = 2007
    s = scramble_date(t, p)
    c1 = (((t.year >> 2) & 0x3fc0) + s) % 25 + 'a'
    c2 = (t.month + s) % 10 + 'a'
    c3 = ((t.year & 0xff) + s) % 25 + 'a'
    if t.day * 2 < '0' ||  t.day * 2 > '9':
        c4 = (t.day * 2) % 25 + 'a'
    else:
        c4 = t.day % 10 + '1'
    return c1 + 'h' + c2 + c3 + 'x' + c4 +
        suffix[t.month - 1]
```

Listing 1: Torpig daily domain generation algorithm.

In practice, the Torpig controllers registered the weekly .com domain and, in a few cases, the corresponding .net domain, for backup purposes. However, they did not register all the weekly domains in advance, which was a critical factor in enabling our hijacking.

The use of domain flux in botnets has important consequences in the arms race between botmasters and defenders. From the attacker's point of view, domain flux is yet another technique to potentially improve the resilience of the botnet against take-down attempts. More precisely, in the event that the current rendezvous point is taken down, the botmasters simply have to register the next domain in the domain list to regain control of their botnet. On the contrary, to the defender's advantage, domain flux opens up the possibility of sinkholing (or "hijacking") a botnet, by registering an available domain that is generated by the botnet's DGAs and returning an answer that is a valid C&C response (to keep bots from switching over to the next domain in the domain list). As we mentioned, Torpig allowed both of these actions: C&C domain names were available for registration, and it was possible to forge valid C&C responses.

The feasibility of these sinkholing attacks depends not only on technical means (e.g., the ability to reverse engineer the botnet protocol and to forge a valid C&C server's response), but also on economic factors, in particular the cost of registering a number of domains sufficient to make the sinkholing effective. Since domain registration comes at a price (currently, from about $5 to $10 per year per .com and .net domain name), botmasters could prevent attacks against domain flux by making them economically infeasible, for example, by forcing defenders to register a disproportionate number of names. Unfortunately, this is a countermeasure that is already in use. Newer variants of Conficker generate 50,000 domains per day and introduce non-determinism in their generation algorithm [32]. Taking over all the domains generated by Conficker at market prices would cost between $91.3 million and $182.5 million per year. Furthermore, the domain flux arms race is clearly in favor of the malware authors. Generating thousands more domains requires an inexpensive modification to the bot code base, while registering them costs time and money.

In short, the idea of combating domain flux by simply acquiring more domains is clearly not scalable in the long term, and new approaches are needed to tilt the balance away from the botmasters. In particular, the security community should build a stronger relationship with registrars. Registrars, in fact, are the entity best positioned to mitigate malware that relies on DNS (including domain flux), but, with few exceptions, they often lack the resources, incentives, or culture to deal with the security issues associated with their roles. In addition, rogue registrars (those known to be a safe haven for the activity of cyber-criminals) should lose their accreditation. While processes exist to terminate registrar accreditation agreements (a recent case involved the infamous EstDomains registrar [2]), they should be streamlined and used more promptly.

4. TAKING CONTROL OF THE BOTNET

In this section, we describe in more detail how we obtained control over the Torpig botnet. We registered domains that bots would resolve and setup a server to which bots would connect to find their C&C. Moreover, we present our data collection and hosting infrastructure and review a timeline of events during our period of control.

The behavior of the botmasters was to not register many of the future Torpig C&C domains in advance. Therefore, we were able to register the .com and .net domains that were to be used by the botnet for three consecutive weeks from January 25th, 2009 to February 15th, 2009. However, on February 4th, 2009, the Mebroot controllers distributed a new Torpig binary that updated the domain algorithm. This ended our control prematurely after ten days. Mebroot domains, in fact, allow botmasters to upgrade, remove, and install new malware components at any time, and are tightly controlled by the criminals. It is unclear why the controllers of the Mebroot botnet did not update the Torpig domain algorithm sooner to thwart our sinkholing.

4.1 Sinkholing Preparation

We purchased service from two different hosting providers that are well-known to be unresponsive to abuse complaints, and we registered our .com and .net domains with two different registrars. This provided redundancy so that if one domain registrar or hosting provider suspended our account, we would be able to maintain control of the botnet. This proved to be useful when our .com domain was suspended on January 31, 2009 due to an abuse complaint. Fortunately, we owned the backup .net domain and were able to continue our collection unabated during this period until we could get our primary domain reinstated.

On our machines, we set up an Apache web server to receive and log bot requests, and we recorded all network traffic[1]. We then automated the process of downloading the data from our hosting providers. Once a data file was downloaded, we removed it from the server on the hosting provider. Therefore, if our servers were compromised, an attacker would not have access to any historical data. During the ten days that we controlled the botnet, we collected over 8.7GB of Apache log files and 69GB of pcap data.

We expected infected machines to connect to us on January 25th, which was the day when bots were supposed to switch to the first weekly domain name that we owned. However, on January 19th, when we started our collection, we instantly received HTTP requests from 359 infected machines. This was almost a week before the expected time. After analyzing the geographical distribution of these machines and the data they were sending, we concluded that these were probably systems that had their clock set incorrectly.

[1]All the collected traffic was encrypted using 256-bit AES.

4.2 Data Collection Principles

During our collection process, we were very careful with the information that we gathered and with the commands that we provided to infected hosts. We operated our C&C servers based on previously established legal and ethical principles [3]. In particular, we protected the victims according to the following:

PRINCIPLE 1. *The sinkholed botnet should be operated so that any harm and/or damage to victims and targets of attacks would be minimized.*

PRINCIPLE 2. *The sinkholed botnet should collect enough information to enable notification and remediation of affected parties.*

There were several preventative measures that were taken to ensure Principle 1. In particular, when a bot contacted our server, we always replied with an `okn` message and never sent it a new configuration file. By responding with `okn`, the bots remained in contact only with our servers. If we had not replied with a valid Torpig response, the bots would have switched over to the .biz domains, which had already been registered by the criminals. Although we could have sent a blank configuration file to potentially remove the web sites currently targeted by Torpig, we did not do so to avoid unforeseen consequences (e.g., changing the behavior of the malware on critical computer systems, such as a server in a hospital). We also did not send a configuration file with a different HTML injection server IP address for the same reasons. To notify the affected institutions and victims, we stored all the data that was sent to us, in accordance with Principle 2, and worked with ISPs and law enforcement agencies, including the United States Department of Defense (DoD) and FBI Cybercrime units, to assist us with this effort. This cooperation also led to the suspension of the current Torpig domains owned by the cyber-criminals.

5. BOTNET ANALYSIS

As mentioned previously, we have collected almost 70GB of data over a period of ten days. The wealth of information that is contained in this data set is remarkable. In this section, we present the results of our data analysis and important insights into the size of botnets and their victims.

5.1 Data Collection and Format

All bots communicate with the Torpig C&C through HTTP POST requests. The URL used for this request contains the hexadecimal representation of the bot identifier and a submission header. The body of the request contains the data stolen from the victim's machine, if any. The submission header and the body are encrypted using the Torpig encryption algorithm (base64 and XOR). The bot identifier (a token that is computed on the basis of hardware and software characteristics of the infected machine) is used as the symmetric key and is sent in the clear.

After decryption, the *submission header* consists of a number of key-value pairs that provide basic information about the bot. More precisely, the header contains the time stamp when the configuration file was last updated (`ts`), the IP address of the bot or a list of IPs in case of a multi-homed machine (`ip`), the port numbers of the HTTP and SOCKS proxies that Torpig opens on the infected machine (`hport` and `sport`), the operating system version and locale (`os` and `cn`), the bot identifier (`nid`), and the build and version number of Torpig (`bld` and `ver`). Figure 3 shows a sample of the header information sent by a Torpig bot.

The request body consists of zero or more *data items* of different types, depending on the information that was stolen. Table 1 shows

```
POST /A15078D49EBA4C4E/qxoT4B5uUFFqw6c35AKDYFpdZHdKLCNn...AaVpJGoSZG1at6E0AaCxQg6nIGA

ts=1232724990&ip=192.168.0.1:&sport=8109&hport=8108&os=5.1.2600&cn=United%20States&
                        nid=A15078D49EBA4C4E&bld=gnh5&ver=229
```

Figure 3: Sample URL requested by a Torpig bot (top) and the corresponding, unencrypted submission header (bottom).

```
[gnh5_229]
[MSO2002-MSO2003:pop.smith.com:John Smith:
    john@smith.com]
[pop3://john:smith@pop.smith.com:110]
[smtp://:@smtp.smith.com:25]
```

```
[gnh5_229]
POST /accounts/LoginAuth
Host:  www.google.com
POST_FORM:
Email=test@gmail.com
Passwd=test
```

Figure 4: Sample data sent by a Torpig bot: a mailbox account on the left, a form data item on the right.

Data Type	Data Items (#)
Mailbox account	54,090
Email	1,258,862
Form data	11,966,532
HTTP account	411,039
FTP account	12,307
POP account	415,206
SMTP account	100,472
Windows password	1,235,122

Table 1: Data items sent to our C&C server by Torpig bots.

the different data types that we observed during our monitoring. In particular, *mailbox account* items contain the configuration information for email accounts, i.e., the email address associated with the mailbox and the credentials required to access the mailbox and to send emails from it. Torpig obtains this information from email clients, such as Outlook, Thunderbird, and Eudora. *Email* items consist of email addresses, which can presumably be used for spam purposes. According to [45], Torpig initially used spam emails to propagate, which may give another explanation for the botmasters' interest in email addresses. *Form data* items contain the content of HTML forms submitted via POST requests by the victim's browser. More precisely, Torpig collects the URL hosting the form, the URL that the form is submitted to, and the name, value, and type of all form fields. These data items frequently contain the usernames and passwords required to authenticate with web sites. Notice that credentials transmitted over HTTPS are not safe from Torpig, since Torpig can access them before they are encrypted by the SSL layer (by hooking appropriate library functions). *HTTP account*, *FTP account*, *POP account*, and *SMTP account* data types contain the credentials used to access web sites, FTP, POP, and SMTP accounts, respectively. Torpig obtains this information by exploiting the password manager functionality provided by most web and email clients. SMTP account items also contain the source and destination addresses of emails sent via SMTP. Finally, the *Windows password* data type is used to transmit Windows passwords and other uncategorized data elements. Figure 4 shows a sample of the data items sent by a Torpig bot.

5.2 Botnet Size

In this section, we address the problem of determining the size of the Torpig botnet. More precisely, we will be referring to two definitions of a botnet's size as introduced by Rajab et al. [34]: the botnet's *footprint*, which indicates the aggregated total number of machines that have been compromised over time, and the botnet's *live population*, which denotes the number of compromised hosts that are simultaneously communicating with the C&C server.

The size of botnets is a hotly contested topic, and one that is widely, and sometimes incorrectly, reported in the popular press [7, 13, 26–28, 30]. Several methods have have been proposed in the past to estimate the size of botnets. These approaches are modeled after the characteristics of the botnet under study and vary along different axes, depending on whether they have access to direct traces of infected machines [6] or have to resort to indirect measurements [11, 34, 35, 37], whether they have a complete or partial view of the infected population, and, finally, whether individual bots are identified by using a network-level identifier (typically, an IP address) or an application-defined identifier (such as a bot ID).

In particular, we briefly compare our measurement technique to those described by Rajab et al. [34] and Kanich et al. [23], who have discussed in detail the methodological aspects of measuring a botnet's size.

Rajab et al. focus on IRC-based botnets. They propose to query DNS server caches to estimate the number of bots that resolved the name of a C&C server and to infiltrate IRC C&C channels with trackers that record the channel activity, in particular, the IDs of channel users. Both methods rely on indirect measurements of bot traffic and are based on active querying and probing. DNS cache querying is partial, since, in its basic form, it only determines if a network contains infected bots, while IRC monitoring can potentially reveal all the bots that connect to a given channel. Finally, the authors observe that IRC identifiers (i.e., nicknames) were found to overestimate the actual size of the botnet,

Kanich et al. focus on P2P botnets. In particular, they measure the size of the Storm network by active probing and crawling the Overnet distributed hash table (DHT). They confirm that the Storm botnet is not ideal for measuring its footprint and live population due to many factors such as protocol aliasing between infected and non-infected Overnet hosts and adversarial aliasing where nodes purposely poison the network to disrupt or impair its operation. The authors also caution that the application IDs used in Overnet were not a good bot identifier, due to a bug in the way they were generated by Storm.

In comparison to these studies, the Torpig C&C's architecture provides an advantageous perspective to measure the botnet's size. In fact, since we centrally and directly observed *every* infected machine that normally would have connected to the botmaster's server during the ten days that we controlled the botnet, we had a complete view of the machines belonging to the botnet. In addition, our collection methodology was entirely passive and, thus, it avoided the problem of active probing that may have otherwise polluted the network that was being measured. Finally, Torpig generates and

transmits unique and persistent IDs that make for good identifiers of infected machines.

In the next section we discuss the characteristics of the botnet that enabled us to determine an overall range for the number of infected machines. We will then compare different methodologies to count the Torpig botnet's footprint and live population.

5.2.1 Counting Bots by nid

As a starting point to estimate the botnet's footprint, we analyzed the `nid` field that Torpig sends in the submission header. Our hypothesis was that this value was unique for each machine and remained constant over time, and that, therefore, it would provide an accurate method to uniquely identify each bot.

By reverse engineering the Torpig binary, we were able to reconstruct the algorithm used to compute this 8-byte value. In particular, the algorithm first queries the primary SCSI hard disk for its model and serial numbers. If no SCSI hard disk is present, or retrieving the disk information is unsuccessful, it will then try to extract the same information from the primary physical hard disk drive (i.e., IDE or SATA). The disk information is then used as input to a hashing function that produces the final `nid` value. If retrieving hardware information fails, the `nid` value is obtained by concatenating the hard-coded value of 0xBAD1D222 with the Windows volume serial number.

In all cases, the `nid` depends on (software or hardware) characteristics of the infected machine's hard disk. Therefore, it does not change, unless the hard disk is replaced (in which case the machine would no longer be infected), or the user manually changes the system's volume serial number (which requires special tools and is not likely to be done by casual users). This gave us confidence that the `nid` remains constant throughout the life of an infected machine.

We then attempted to validate whether the `nid` is unique for each bot. Therefore, we correlated this value with the other information provided in the submission header and bot connection patterns to our server. In particular, we were expecting that all submissions with a specific `nid` would report the same values for the `os`, `cn`, `bld`, and `ver` fields. Unfortunately, we found 2,079 cases for which this assumption did not hold.

Therefore, we conclude that counting unique `nid`s underestimates the botnet's footprint. As a reference point, between Jan 25, 2009 and February 4, 2009, 180,835 `nid` values were observed.

5.2.2 Counting Bots by Submission Header Fields

As a more accurate method to identify infected machines, we used the `nid`, `os`, `cn`, `bld`, and `ver` values from the submission header that Torpig bots send. As we have seen, the `nid` value is mostly unique among bots, and the other fields help distinguishing different machines that have the same `nid`. In particular, the `os` (OS version number) and `cn` (locale information) fields are determined by using the system calls `GetVersionEx` and `GetLocaleInfo`, respectively, and do not change unless the user modifies the locale information on her computer or changes her OS. The values of the `bld` and `ver` fields are hard-coded into the Torpig binary.

We decided not to use the `ts` field (time stamp of the configuration file), since its value is determined by the Torpig C&C that distributed the configuration file and not by characteristics of the bot. Also, we discarded the `ip` field, since it could change depending on DHCP and other network configurations, and the `sport` and `hport` fields, which specify the proxy ports that Torpig opens on the local machine, because they could change after a reboot.

By counting unique tuples from the Torpig headers consisting of (`nid`, `os`, `cn`, `bld`, `ver`), we estimate that the botnet's footprint for the ten days of our monitoring consisted of 182,914 machines.

5.2.3 Identifying Probers and Researchers

Finally, we wanted to identify security researchers and other curious individuals who probed our botnet servers. These do not correspond to actual victims of the botnet and, therefore, we would like to identify them and subtract them from the total botnet size.

We used two heuristics to identify probers and (likely) security researchers. First, we observed that the `nid` values generated by infected clients running in virtual machines is constant. This is because the `nid` depends essentially on physical characteristics of the hard disk, and, by default, virtual machines provide virtual devices with a fixed model and serial number. Since virtual machines are often used by researchers to study malware in a contained environment, we assume that these bots in reality correspond to researchers studying the Torpig malware. In particular, we were able to determine the `nid` values generated on a standard configuration of the VMware and QEMU virtual machines and we found 40 hosts using these values. Second, we identified hosts that send invalid requests to our C&C server (i.e., requests that cannot be generated by Torpig bots). For example, these bots used the GET HTTP method in requests where a real Torpig bot would use the POST method. Using this approach, we discounted another 74 hosts. We further ignored background noise, such as scanning of our web server and traffic from search engine bots. After subtracting probers and researchers, our final estimate of the botnet's footprint is 182,800 hosts.

5.2.4 Botnet Size vs. IP Count

It is well-known that, due to network effects such as DHCP churn and NAT, counting the number of infected bots by counting the unique IP addresses that connect to the botnet's C&C server is problematic [34]. In this section, we examine the relationship between the botnet size and the IP counts in more detail.

As we discussed, during our ten days of monitoring, we observed 182,800 bots. In contrast, during the same time, 1,247,642 unique IP addresses contacted our server. Taking this value as the botnet's footprint would overestimate the actual size by an order of magnitude. We further analyzed the difference between IP count and the actual bot count by examining their temporal characteristics. In particular, Figure 5 displays the number of unique IP addresses observed during the ten days that we were in control of the Torpig C&C. After the initial spike when the bots started to contact our server, there was a consistent diurnal pattern of unique IP addresses with an average of 4,690 new IPs per hour. In contrast, the average number of new bots observed was 705 per hour, with a very rapid drop-off after the first peak, as shown in Figure 6. Therefore, the number of cumulative new IP addresses that we saw over time increased linearly, as shown in Figure 7. On the other hand, the aggregate number of new bots observed decayed quickly. Figure 8 shows that more than 75% of all new Torpig bots during the ten-day interval were observed in the first 48 hours.

While the aggregate number of total unique IP addresses distorts the botnet's footprint and live population, the number of IP addresses can be used to closely approximate the botnet's size using other metrics. The median and average size of Torpig's live population was 49,272 and 48,532, respectively. The live population fluctuates periodically, where the peaks correspond to 9:00am Pacific Standard Time (PST), when the most computers are simultaneously online in the United States and Europe. Conversely, the smallest live population occurs around 9:00pm PST, when more people in the United States and Europe are offline. When we com-

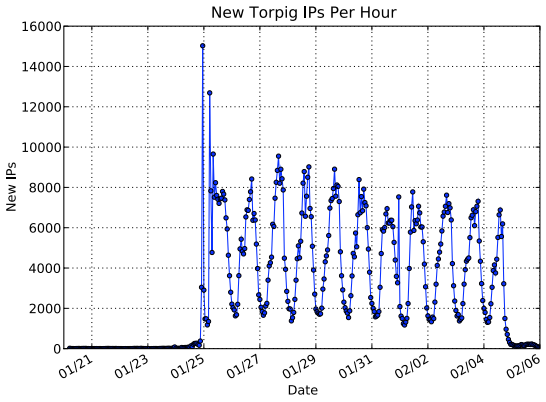


Figure 5: New unique IP addresses per hour.

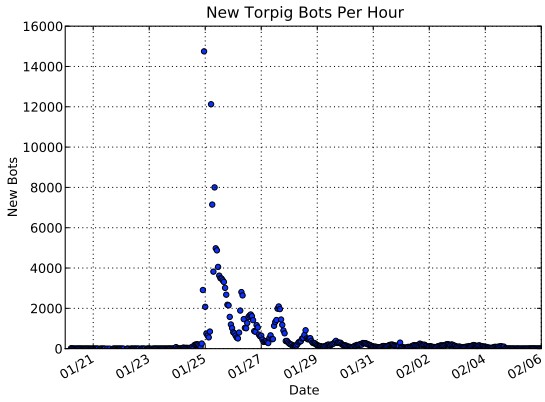


Figure 6: New bots per hour.

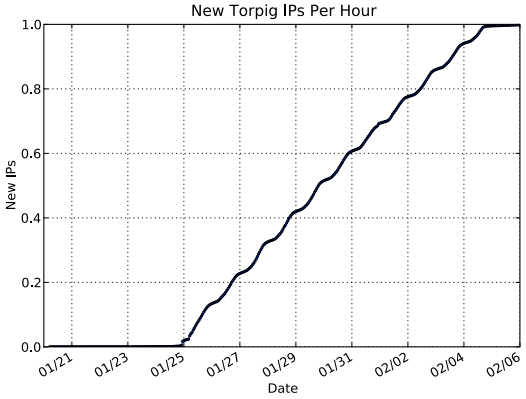


Figure 7: CDF – New unique IP addresses per hour.

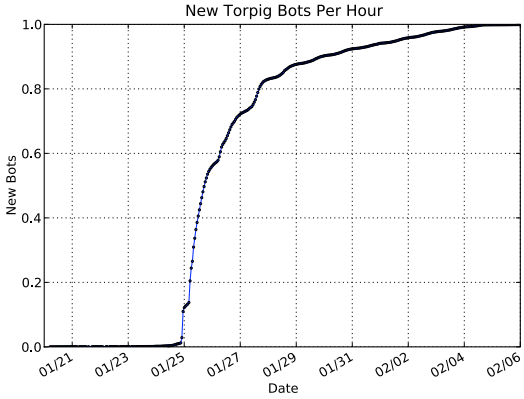


Figure 8: CDF – New bots per hour.

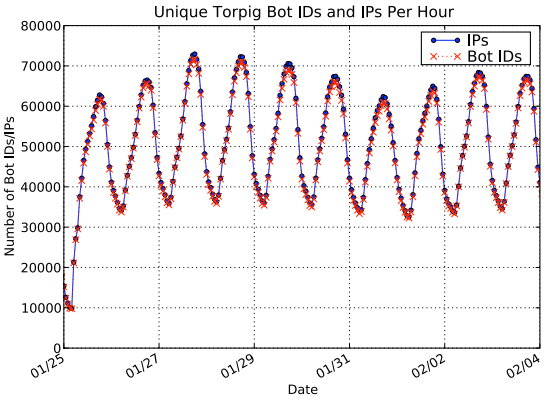


Figure 9: Unique Bot IDs and IP addresses per hour.

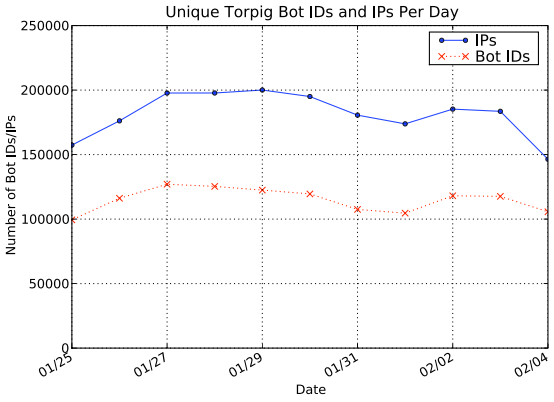


Figure 10: Unique Bot IDs and IP addresses per day.

pare the observed number of unique bot IDs per hour with the number of unique IP addresses, they are virtually identical (as shown in Figure 9). On average, the bot IDs were only 1.3% less than the number of IP addresses per hour. Thus, the number of unique IPs per hour provides a good estimation of the botnet's live population. The similarity between bot IDs and IPs per hour is a consequence of each infected host connecting to the C&C every 20 minutes, which occurs more frequently than the rate of DHCP churn. Hence, the more often a bot connects to the C&C, the more accurate an IP count will be to the live population on an hourly scale. In comparison, the number of IPs per day does not accurately reflect the botnet's live population (as shown in Figure 10), with a difference of 36.5% between IP addresses and bot IDs. The median number and average number of IPs per day during our ten days of controlling the C&C was 182,058 and 179,866 respectively. Interestingly, both of these statistics provide a reasonable approximation to the botnet's footprint in comparison to the bot IDs.

The difference between IP count and the actual bot count can be attributed to DHCP and NAT effects. In networks using the DHCP protocol (or connecting through dial-up lines), clients (machines on the network) are allocated an address from a pool of available IP addresses. The allocation is often dynamic, that is, a client is not guaranteed to always be assigned the same IP address. This can inflate the number of observed IP addresses at the botnet C&C server. Short leases (the length of time for which the allocation is valid) can further magnify this effect. This phenomenon was very common during our monitoring. In fact, we identified the presence of ISPs that rotate IP addresses so frequently that almost every time that an infected host on their network connected to us, it had a new IP address. In one instance, a single host had changed IP addresses 694 times in just ten days! In some cases, the same host was associated with different IP addresses on the same autonomous systems, but different class B /16 subnets. We observed this DHCP churn on several different networks with the most common being, in descending order: Deutsche Telekom, Verizon, and BellSouth. Overall, there were 706 different machines that were seen with more than one hundred unique IP addresses. At this point, we can only speculate why these ISPs recycle IP addresses so frequently.

Country	IP Addresses (Raw #)	Bot IDs	DHCP Churn Factor
US	158,209	54,627	2.90
IT	383,077	46,508	8.24
DE	325,816	24,413	13.35
PL	44,117	6,365	6.93
ES	31,745	5,733	5.54
GR	45,809	5,402	8.48
CH	30,706	4,826	6.36
UK	21,465	4,792	4.48
BG	11,240	3,037	3.70
NL	4,073	2,331	1.75
Other	180,070	24,766	7.27
Totals:	1,247,642	182,800	6.83

Table 2: Top 10 infected hosts by country.

Furthermore, by comparing the number of bots we observed and their IP addresses, we can determine the effect of DHCP churn at a country level. Interestingly, the IP address count significantly overestimates the infection count in some countries, because the ISPs in those regions recycle IP addresses more often in comparison to others as shown in Table 2. For instance, a naïve estimate per country would consider Italy and Germany to have the largest number of infections. However, the ISPs in those countries assign IP addresses *much* more frequently than their U.S. counterparts. In fact, Germany had less than half the number of infected hosts, yet double the number of IP address connections. Furthermore, the ratio of IPs to hosts in Germany was four times higher than that of the United States. Because Torpig spreads through drive-by-download web sites, we believe the clustering by country reflects that most of the malicious sites use English, Italian, or German, since these are the top affected countries.

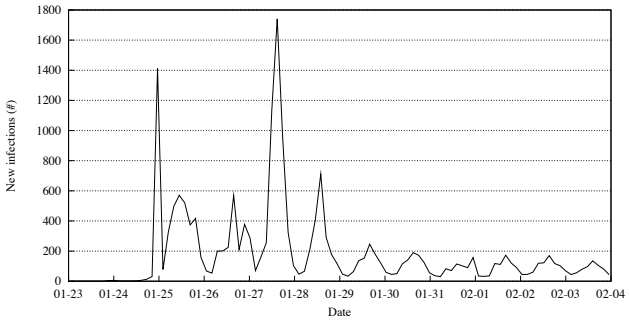


Figure 11: New infections over time.

The information provided in the Torpig headers also allows us to estimate the impact of NAT, which is commonly used to enable shared Internet access for an entire private network through a single public access (masquerading). This technique reduces the number of IPs observed at the C&C server, since all the infected machines in the masqueraded network would count as one. By looking at the IP addresses in the Torpig headers we are able to determine that 144,236 (78.9%) of the infected machines were behind a NAT, VPN, proxy, or firewall. We identified these hosts by using the non-publicly routable IP addresses listed in RFC 1918: 10/8, 192.168/16, and 172.16-172.31/16. We observed 9,336 distinct bots for 2,753 IP addresses from these infected machines on private networks. Therefore, if the IP address count was used to determine the number of hosts it would underestimate the infection count by a factor of more than 3 times.

5.2.5 New Infections

The Torpig submission header provides the time stamp of the most recently received configuration file. We leveraged this fact to approximate the number of machines newly infected during the period of observation by counting the number of distinct victims whose initial submission header contains a time stamp of 0. Figure 11 shows the new infections over time. In total, we estimate that there were 49,294 new infections while the C&C was under our control. New infections peaked on the 25th and the 27th of January. We can only speculate that, on those days, a popular web site was compromised and redirected its visitors to the drive-by-download servers controlled by the botmasters.

5.3 Botnet as a Service

An interesting aspect of the Torpig botnet is that there are indications that different groups would be dividing (and profiting from) the data it steals. Torpig DLLs are marked with a *build* type represented by the `bld` field in the header. This value is set during the drive-by download (the build type is included in the URL that triggers the download) and remains the same during the entire life cycle of an infection. The build type does not seem to indicate different feature sets, since different Torpig builds behave in the same way. However, Torpig transmits its build type in all communications with the C&C server, and, in particular, includes it in both the submission header (as the `bld` parameter) and in each data item contained in a submission body (for example, in Figure 3 the build

Country	Institutions (#)	Accounts (#)
US	60	4,287
IT	34	1,459
DE	122	641
ES	18	228
PL	14	102
Other	162	1,593
Total	410	8,310

Table 3: Accounts at financial institutions stolen by Torpig.

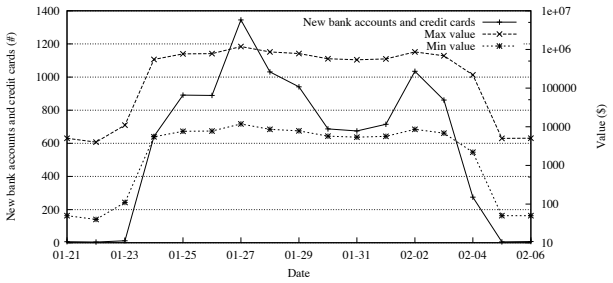


Figure 12: The arrival rate of financial data.

type was `gnh5`). Therefore, the most convincing explanation of the build type is that it denotes different "customers" of the Torpig botnet, who, presumably, get access to their data in exchange for a fee. If correct, this interpretation would mean that Torpig is actually used as a "malware service", accessible to third parties who do not want or cannot build their own botnet infrastructure.

During our study, we observed 12 different values for the `bld` parameter: `dxtrbc`, `eagle`, `gnh1`, `gnh2`, `gnh3`, `gnh4`, `gnh5`, `grey`, `grobin`, `grobin1`, `mentat`, and `zipp`. Not all builds contribute equally to the amount of data stolen. The most active versions are `dxtrbc` (5,432,528 submissions), `gnh5` (2,836,198), and `mentat` (1,582,547).

6. THREATS AND DATA ANALYSIS

In this section, we will discuss the threats that Torpig poses and will turn our attention to the actual data that infected machines sent to our C&C server. We will see that Torpig creates a considerable potential for damage due not only to the shear volume of data it collects, but also to the amount of computing resources the botnet makes available.

6.1 Financial Data Stealing

Consistent with the past few years' shift of malware from a for-fun (or notoriety) activity to a for-profit enterprise [10, 15], Torpig is specifically crafted to obtain information that can be readily monetized in the underground market. Financial information, such as bank accounts and credit card numbers, is particularly sought after. For example, the typical Torpig configuration file lists roughly 300 domains belonging to banks and other financial institutions that will be the target of the "man-in-the-browser" phishing attacks described in Section 2.

Table 3 reports the number of accounts at financial institutions (such as banks, online trading, and investment companies) that were stolen by Torpig and sent to our C&C server. In ten days, Torpig obtained the credentials of 8,310 accounts at 410 different institutions. The top targeted institutions were PayPal (1,770 accounts), Poste Italiane (765), Capital One (314), E*Trade (304), and Chase (217). On the other end of the spectrum, a large number of companies had only a handful of compromised accounts (e.g., 310 had ten or less). The large number of institutions that had been breached made notifying all of the interested parties a monumental effort. It is also interesting to observe that 38% of the credentials stolen by Torpig were obtained from the password manager of browsers, rather than by intercepting an actual login session. It was possible to infer that number because Torpig uses different data formats to upload stolen credentials from different sources.

Another target for collection by Torpig is credit card data. Using a credit card validation heuristic that includes the Luhn algorithm and matching against the correct number of digits and numeric prefixes of card numbers from the most popular credit card companies, we extracted 1,660 unique credit and debit card numbers from our collected data. Through IP address geolocation, we surmise that 49% of the card numbers came from victims in the US, 12% from Italy, and 8% from Spain, with 40 other countries making up the balance. The most common cards include Visa (1,056), MasterCard (447), American Express (81), Maestro (36), and Discover (24).

While 86% of the victims contributed only a single card number, others offered a few more. Of particular interest is the case of a single victim from whom 30 credit card numbers were extracted. Upon manual examination, we discovered that the victim was an agent for an at-home, distributed call center. It seems that the card numbers were those of customers of the company that the agent was working for, and they were being entered into the call center's central database for order processing.

Quantifying the value of the financial information stolen by Torpig is an uncertain process because of the characteristics of the underground markets where it may end up being traded. A report by Symantec [43] indicated (loose) ranges of prices for common goods and, in particular, priced credit cards between $0.10–$25 and bank accounts from $10–$1,000. If these figures are accurate, in ten days of activity, the Torpig controllers may have profited anywhere between $83K and $8.3M.

Furthermore, we wanted to determine the rate at which the botnet produces *new* financial information for its controllers. Clearly, a botnet that generates all of its value in a few days and later only recycles stale information is less valuable than one where fresh data is steadily produced. Figure 12 shows the rate at which new bank accounts and credit card numbers were obtained during our monitoring period. In the ten days when we had control of the botnet, new data was continuously stolen and reported by Torpig bots.

6.2 Proxies

As we mentioned previously, Torpig opens two ports on the local machine, one to be used as a SOCKS proxy, the other as an HTTP proxy. 20.2% of the machines we observed were publicly accessible. Their proxies, therefore, could be easily leveraged by miscreants to, for example, send spam or navigate anonymously. In particular, we wanted to verify if spam was sent through machines in the Torpig botnet. We focused on the 10,000 IPs that contacted us most frequently. These, arguably, correspond to machines that are available for longer times and that are, thus, more likely to be used by the botmasters. We matched these IPs against the ZEN blocklist, a well-known and accurate list of IP addresses linked to spamming, which is compiled by the Spamhaus project [44]. We found that one IP was marked as a verified spam source or spam operation and 244 (2.45%) were flagged as having open proxies that are used for spam purposes or being infected with spam-related malware. While we have no evidence that the presence of these IPs

Network Speed	IP Addresses (Raw #)	Bot IDs	DHCP Churn Factor
Cable/DSL	356,428	50,535	7.05
Dial-up	129,493	9,923	13.05
Corporate	40,818	17,217	2.37
Unknown	677,434	105,125	6.44

Table 4: Network speed of infected hosts.

on the ZEN blocklist is a consequence of the Torpig infection, it is clear that Torpig has the potential to drag its victims into a variety of malicious activities. Furthermore, since most IPs are "clean", they can be used for spamming, anonymous navigation, or other dubious enterprises.

6.3 Denial-of-Service

To approximate the amount of aggregate bandwidth among infected hosts, we mapped the IP addresses to their network speed, using the ip2location[2] database. This information is summarized in Table 4. Unfortunately the database does not contain records for about two-thirds of the IP addresses, but from the information that it provides, we can see that cable and DSL lines account for 65% of the infected hosts. If we assume the same distribution of network speed for the unknown IP addresses, there is a tremendous amount of bandwidth in the hands of the botmaster, considering that there were more than 70,000 active hosts at peak intervals. In 2008, the median upstream bandwidth in the United States was 435 kbps for DSL connections [42]. Since the United States ranks as one of the slowest in terms of broadband speeds, we will use 435 kbps as a conservative estimate for each bot's upstream bandwidth. Thus, the aggregate bandwidth for the DSL/Cable connections is roughly 17 Gbps. If we further add in corporate networks, which account for 22% of infected hosts, and consider that they typically have significantly larger upstream connections, the aggregate bandwidth is likely to be considerably higher. Hence, a botnet of this size could cause a massive distributed denial-of-service (DDoS) attack.

6.4 Password Analysis

A recent poll conducted by Sophos in March 2009 [41], reported that one third of 676 Internet users neglect the importance of using strong passwords and admitted that they reused online authentication credentials across different web services. While it is reasonable to trust the results of a poll, it is also important to cross-validate these results, as people may not always report the truth. Typically, this validation task relies on the presence of ground truth, which is generally missing or very hard to obtain.

Interesting enough, our effort to take over the Torpig botnet over a ten-days period offered us the rare opportunity to obtain the necessary ground truth to validate the results of the Sophos poll. The benefits of the credential analysis we performed are twofold. First, it is possible to rely on real data, i.e., data that had been actually collected, and not on user-provided information, which could be fake. Second, the data corpus provided by the Torpig-infected machines was two orders of magnitude bigger than the one used in the Sophos poll, and results derived from a large data corpus are usually less prone to outliers and express trends in a better way than those performed on a smaller one.

Torpig bots stole 297,962 unique credentials (username and password pairs), sent by 52,540 different Torpig-infected machines, over the period we controlled the botnet. The stolen credentials were discovered as follows. For each infected host $\mathcal{H}$, we retrieved all the unique username and password pairs c submitted by $\mathcal{H}$. Afterward, the number w_c of distinct web services where a credential c was used was obtained. Finally, we concluded that c had been reused across w_c different web services, if w_c was greater than or equal to 2.

Our analysis found that almost 28% of the victims reused their credentials for accessing 368,501 web sites. While this percentage is slightly lower than the results reported in the poll conducted by Sophos, it is close enough to confirm and validate it.

In addition to checking for credential reuse, we also conducted an experiment to assess the strength of the 173,686 unique passwords discovered in the experiment above. To this end, we created a UNIX-like password file to feed John the Ripper, a popular password cracker tool [31]. The results are presented in Figure 13.

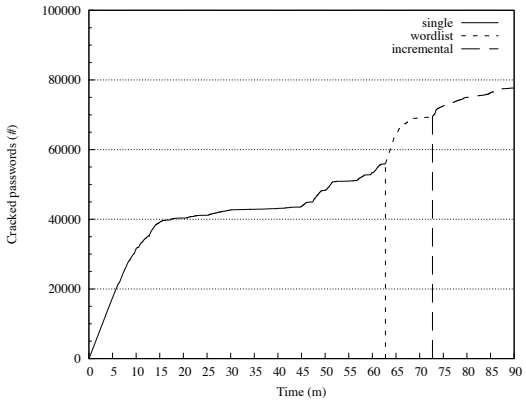


Figure 13: Number of passwords cracked in 90 minutes by the John the Ripper password cracker tool. Vertical lines indicate when John switches cracking mode. The first vertical line represents the switching from simple transformation techniques ("single" mode) to wordlist cracking, the second from wordlist to brute-force ("incremental").

About 56,000 passwords were recovered in less than 65 minutes by using permutation, substitution, and other simple replacement rules used by the password cracker (the "single" mode). Another 14,000 passwords were recovered in the next 10 minutes when the password cracker switched modes to use a large wordlist. Thus, in less than 75 minutes, more than 40% of the passwords were recovered. 30,000 additional passwords were recovered in the next 24 hours by brute force (the "incremental" mode).

7. RELATED WORK

Other analyses of both Mebroot and Torpig have been done [9, 12, 25]. These primarily focus on the Master Boot Record (MBR) overwriting rootkit technique employed by Mebroot. We complement this work, since the focus of our analysis has been on the Torpig botnet.

Torpig utilizes a relatively new strategy for locating its C&C servers, which we refer to as *domain flux*. Analyses of other bot families like Kraken/Bobax [1], Srizbi [46], and more recently, Conficker [32], have revealed that the use of the domain flux technique for bot coordination is on the rise. We present domain flux in detail and discuss its strengths and weaknesses, and we propose several remediation strategies.

Botnet takeover as an analysis and defense strategy have been considered elsewhere. Kanich *et al.* infiltrated the Storm botnet by impersonating proxy peers in the overlay network. They demonstrated their control by rewriting URLs in the spam sent by the

[2]http://www.ip2location.com

bots [22]. Recent efforts to disrupt the Conficker botnet have focused on sinkholing future rendezvous domains in order to disable the botmaster's ability to update the infected machines [19]. Our takeover of Torpig is closest in spirit to the latter effort, as we also took advantage of the shortcomings of using domain flux for C&C.

Determining the size of a botnet is difficult. Many studies have used the number of unique IP addresses to estimate the number of compromised hosts [34]. Recently, Conficker has been reported to have infected between one and ten million machines using this heuristic [32]. The Storm botnet's size was approximated by crawling the Overnet distributed hash table (DHT) and counting DHT identifiers and IP address pairs [18]. We believe many of these studies overestimate the actual bot population size for the reasons we detailed previously. On the contrary, we have provided a detailed discussion of how we determine the size of the Torpig botnet.

There has been work focused on understanding the information harvested by malware. For example, Holz *et al.* analyzed data from 70 dropzone servers containing information extracted from keyloggers [16]. Also, a Torpig server was seized in 2008, resulting in the recovery of 250,000 stolen credit and debit cards and 300,000 online bank account login credentials [38]. Furthermore, Franklin *et al.* classified and assessed the value of compromised credentials for financial and other personal information that is bought and sold in the underground Internet economy [10]. Unlike these studies, we analyzed live data that was sent directly to us by bots. This allows us to gain further insights, such as the timing relationships between events.

8. CONCLUSIONS

In this paper, we present a comprehensive analysis of the operations of the Torpig botnet. Controlling hundreds of thousands of hosts that were volunteering Gigabytes of sensitive information provided us with the unique opportunity to understand both the characteristics of the botnet victims and the potential for profit and malicious activity of the botnet creators.

There are a number of lessons learned from the analysis of the data we collected, as well as from the process of obtaining (and losing) the botnet. First, we found that a naïve evaluation of botnet size based on the count of distinct IPs yields grossly overestimated results (a finding that confirms previous, similar results). Second, the victims of botnets are often users with poorly maintained machines that choose easily guessable passwords to protect access to sensitive sites. This is evidence that the malware problem is fundamentally a *cultural* problem. Even though people are educated and understand well concepts such as the physical security and the necessary maintenance of a car, they do not understand the consequences of irresponsible behavior when using a computer. Therefore, in addition to novel tools and techniques to combat botnets and other forms of malware, it is necessary to better educate the Internet citizens so that the number of potential victims is reduced. Third, we learned that interacting with registrars, hosting facilities, victim institutions, and law enforcement is a rather complicated process. In some cases, simply identifying the point of contact for one of the registrars involved required several days of frustrating attempts. We are sure that we have not been the first to experience this type of confusion and lack of coordination among the many pieces of the botnet puzzle. However, in this case, we believe that simple rules of behavior imposed by the US government would go a long way toward preventing obviously-malicious behavior.

Acknowledgments

The research was supported by the National Science Foundation, under grant CNS-0831408.

We would like to acknowledge the following people and groups for their help during this project: David Dagon, MELANI/GovCERT.ch[3], and the Malware Domain List community[4].

9. REFERENCES

[1] P. Amini. Kraken Botnet Infiltration. http://dvlabs.tippingpoint.com/blog/2008/04/28/kraken-botnet-infiltration, 2008.

[2] S. Burnette. Notice of Termination of ICANN Registrar Accreditation Agreement. http://www.icann.org/correspondence/burnette-to-tsastsin-28oct08-en.pdf, 2008.

[3] A. Burstein. Conducting Cybersecurity Research Legally and Ethically. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[4] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2006.

[5] D. Dagon, G. Gu, C. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Annual Computer Security Applications Conference (ACSAC)*, 2007.

[6] D. Dagon, C. Zou, and W. Lee. Modeling Botnet Propagation Using Time Zones. In *Symposium on Network and Distributed System Security*, 2006.

[7] Finjan. How a cybergang operates a network of 1.9 million infected computers. http://www.finjan.com/MCRCblog.aspx?EntryId=2237, 2009.

[8] J. Fink. FBI Agents Raid Dallas Computer Business. http://cbs11tv.com/local/Core.IP.Networks.2.974706.html, 2009.

[9] E. Florio and K. Kasslin. Your computer is now stoned (...again!). *Virus Bulletin*, April 2008.

[10] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *ACM Conference on Computer and Communications Security*, 2007.

[11] F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *European Symposium on Research in Computer Security (ESORICS)*, 2005.

[12] GMER Team. Stealth MBR rootkit. http://www2.gmer.net/mbr/, 2008.

[13] D. Goodin. Superworm seizes 9m pcs, 'stunned' researchers say. http://www.theregister.co.uk/2009/01/16/9m_downadup_infections/, 2009.

[14] P. Guehring. Concepts against Man-in-the-Browser Attacks. http://www2.futureware.at/svn/sourcerer/CAcert/SecureClient.pdf, 2006.

[15] P. Gutmann. The Commercial Malware Industry. In *DEFCON conference*, 2007.

[16] T. Holz, M. Engelberth, and F. Freiling. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. Reihe Informatik TR-2008-006, University of Mannheim, 2008.

[3]Email: cert@melani.admin.ch, URL:http://www.melani.admin.ch/index.html?lang=en

[4]http://www.malwaredomainlist.com/

[17] T. Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Symposium on Network and Distributed System Security*, 2008.

[18] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[19] J. Hruska. Cracking down on Conficker: Kaspersky, OpenDNS join forces. http://arstechnica.com/business/news/2009/02/cracking-down-on-conficker-kaspersky-opendns-join-forces, February 2009.

[20] D. Jackson. Untorpig. http://www.secureworks.com/research/tools/untorpig/, 2008.

[21] B. Kang, E. Chan-Tin, C. Lee, J. Tyra, H. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon, and Y. Kim. Towards complete node enumeration in a peer-to-peer botnet. In *ACM Symposium on Information, Computer & Communication Security (ASIACCS 2009)*, 2009.

[22] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. Spamalytics: An Empirical Analysis of Spam Marketing Conversion. In *ACM Conference on Computer and Communications Security*, 2008.

[23] C. Kanich, K. Levchenko, B. Enright, G. Voelker, and S. Savage. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[24] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *USENIX Workshop on Hot Topics in Understanding Botnet*, 2007.

[25] P. Kleissner. Analysis of Sinowal. http://web17.webbpro.de/index.php?page=analysis-of-sinowal, 2008.

[26] J. Leyden. Conficker botnet growth slows at 10m infections. http://www.theregister.co.uk/2009/01/26/conficker_botnet/, 2009.

[27] J. Leyden. Conficker zombie botnet drops to 3.5 million. http://www.theregister.co.uk/2009/04/03/conficker_zombie_count/, 2009.

[28] R. McMillan. Conficker group says worm 4.6 million strong. http://www.cw.com.hk/content/conficker-group-says-worm-46-million-strong, 2009.

[29] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Usenix Security Symposium*, 2001.

[30] G. Ollmann. Caution Over Counting Numbers in C&C Portals. http://blog.damballa.com/?p=157, 2009.

[31] Openwall Project. John the Ripper password cracker. http://www.openwall.com/john/.

[32] P. Porras, H. Saidi, and V. Yegneswaran. A Foray into Conficker's Logic and Rendezvous Points. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2009.

[33] N. Provos and P. Mavrommatis. All Your iFRAMEs Point to Us. In *USENIX Security Symposium*, 2008.

[34] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My Botnet is Bigger than Yours (Maybe, Better than Yours) : Why Size Estimates Remain Challenging. In *USENIX Workshop on Hot Topics in Understanding Botnet*, 2007.

[35] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *ACM Internet Measurement Conference (IMC)*, 2006.

[36] A. Ramachandran and N. Feamster. Understanding the Network-level Behavior of Spammers. In *ACM SIGCOMM*, 2006.

[37] A. Ramachandran, N. Feamster, and D. Dagon. Revealing Botnet Membership Using DNSBL Counter-Intelligence. In *Conference on Steps to Reducing Unwanted Traffic on the Internet*, 2006.

[38] RSA FraudAction Lab. One Sinowal Trojan + One Gang = Hundreds of Thousands of Compromised Accounts. http://www.rsa.com/blog/blog_entry.aspx?id=1378, October 2008.

[39] S. Saroiu, S. Gribble, and H. Levy. Measurement and Analysis of Spyware in a University Environment. In *Networked Systems Design and Implementation (NSDI)*, 2004.

[40] M. Shields. Trojan virus steals banking info. http://news.bbc.co.uk/2/hi/technology/7701227.stm, 2008.

[41] Sophos. Security at risk as one third of surfers admit they use the same password for all websites, Sophos reports. http://www.sophos.com/pressoffice/news/articles/2009/03/password-security.html, March 2009.

[42] SpeedMatters.org. 2008 Report on Internet Speeds in All 50 States. http://www.speedmatters.org/document-library/sourcematerials/cwa_report_on_internet_speeds_2008.pdf, August 2008.

[43] Symantec. Report on the underground economy. http://www.symantec.com/content/en/us/about/media/pdfs/Underground_Econ_Report.pdf, 2008.

[44] The Spamhaus Project. ZEN. http://www.spamhaus.org/zen/.

[45] VeriSign iDefense Intelligence Operations Team. The Russian Business Network: Rise and Fall of a Criminal ISP. blog.wired.com/defense/files/iDefense_RBNUpdated_20080303.doc, 2008.

[46] J. Wolf. Technical details of Srizbi's domain generation algorithm. http://blog.fireeye.com/research/2008/11/technical-details-of-srizbis-domain-generation-algorithm.html, 2008.

[47] L. Zhuang, J. Dunagan, D. Simon, H. Wang, I. Osipkov, G. Hulten, and J. Tygar. Characterizing botnets from email spam records. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

Improving Network Availability with Protective ReRoute

David Wetherall, Abdul Kabbani*, Van Jacobson, Jim Winget, Yuchung Cheng,
Charles B. Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat
Google

ABSTRACT

We present PRR (Protective ReRoute), a transport technique for shortening user-visible outages that complements routing repair. It can be added to any transport to provide benefits in multipath networks. PRR responds to flow connectivity failure signals, e.g., retransmission timeouts, by changing the FlowLabel on packets of the flow, which causes switches and hosts to choose a different network path that may avoid the outage. To enable it, we shifted our IPv6 network architecture to use the FlowLabel, so that hosts can change the paths of their flows without application involvement. PRR is deployed fleetwide at Google for TCP and Pony Express, where it has been protecting all production traffic for several years. It is also available to our Cloud customers. We find it highly effective for real outages. In a measurement study on our network backbones, adding PRR reduced the cumulative region-pair outage time over TCP with application-level recovery by 63–84%. This is the equivalent of adding 0.4–0.8 "nines" of availability.

CCS CONCEPTS

• **Networks → Data path algorithms**; **Network reliability**; **End nodes**;

KEYWORDS

Network availability, Multipathing, FlowLabel

ACM Reference Format:
David Wetherall, Abdul Kabbani, Van Jacobson, Jim Winget, Yuchung Cheng, Charles B. Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat. 2023. Improving Network Availability with Protective ReRoute. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3603269.3604867

1 INTRODUCTION

Hyperscalers operate networks that support services with billions of customers for use cases such as banking, hospitals and commerce. High network availability is a paramount concern. Since it is impossible to prevent faults in any large network, it is necessary to repair them quickly to avoid service interruptions. The classic repair strategies depend on routing protocols to rapidly shift traffic from failed links and switches onto working paths.

In our experience, the main barrier to high availability in large networks is the non-negligible number of faults for which routing does not quickly restore connectivity. Sometimes routing fails to help due to configuration mistakes, software bugs, or unexpected interactions that are only revealed after a precipitating event, such as a hardware fault. For example, bugs in switches may cause packets to be dropped without the switch also declaring the port down. Or switches may become isolated from their SDN controllers so that part of the network cannot be controlled. And routing or traffic engineering may use the wrong weights and overload links.

While unlikely, these complex faults do occur in hyperscaler networks due to their scale. They result in prolonged outages due to the need for higher-level repair workflows. This has an outsize impact on availability: a single 5 min outage means <99.99% monthly uptime. Qualitatively, outages that last minutes are highly disruptive for customers, while brief outages lasting seconds may not be noticed.

Perhaps surprisingly, routing is insufficient even for the outages that it can repair. Traffic control systems operate at multiple timescales. Fast reroute [3, 4] performs local repair within seconds to move traffic to backup paths for single faults. Routing performs a global repair, taking tens of seconds in very large networks to propagate topology updates, compute new routes, and install them at switches. Finally, traffic engineering responds within minutes to fit demand to capacity by adjusting path weights.

For routing alone to maintain high availability, we need fast reroute to always succeed. Yet a significant fraction of outages only fully recover via global routing and traffic engineering. This is because fast reroute depends on limited backup paths. These paths may not cover the actual fault because they are pre-computed for the Shared Risk Link Group (SRLG) [9] of a planned fault. Backup paths that do survive the fault are less performant than the failed paths they replace. Because the number of paths grows exponentially with path length, there are fewer local options for paths between the switches adjacent to a fault than global end-to-end paths that avoid the same fault. With limited options plus the capacity loss due to the fault, backup paths are easily overloaded.

Our reliability architecture shifts the responsibility for rapid repair from routing to hosts. Modern networks provide the opportunity to do so because they have scaled by adding more links, not only faster links. To add capacity, the links must be disjoint. This leads to multiple paths between pairs of endpoints that can fail independently. Thus the same strategy that scales capacity also adds diversity that can raise reliability.

Hosts typically see bimodal outage behavior: some connections take paths to a destination that are *black holes* which discard packets in the network, while other connections to the same destination take paths that continue to work correctly. This partial loss of connectivity is a consequence of multiple paths plus standard techniques to avoid single points of failure. For example, it is unlikely

*The author contributed to this work while at Google.

ACM SIGCOMM '23, September 10, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3604867

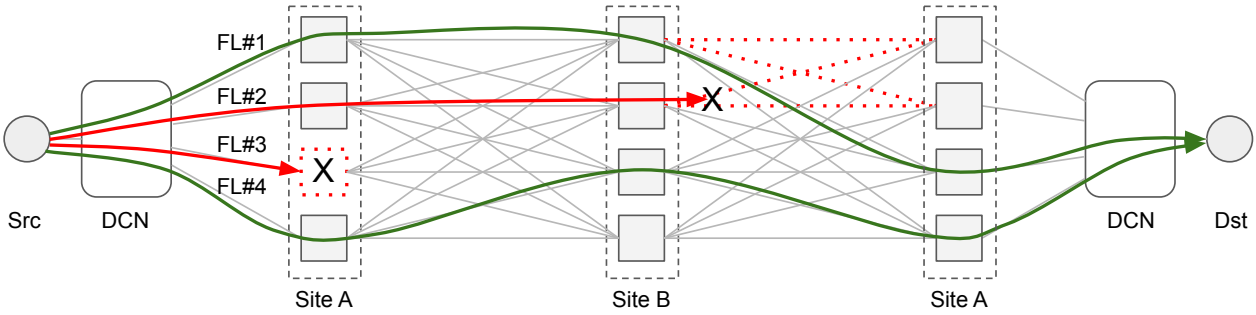


Figure 1: Multipath between hosts can route around faults shown in dotted red.

that a fault will impact geographically distinct fiber routes, or all shards of a network entity that is partitioned for reliability.

To restore connectivity, hosts must avoid failed paths and use working ones. This poses a dilemma because the IPv4 architecture does not let hosts influence the network path of a flow. Instead, with the rise of multiple paths, Equal-Cost Multi-Path (ECMP) [24, 48] routing at switches uses transport identifiers to spread flows across paths. This design ties a connection to a path. Fortunately, now that we have migrated Google networks to IPv6, we can leverage the IPv6 FlowLabel [2, 34] in its intended architectural role to influence the path of connections.

With Protective ReRoute (PRR), switches include the FlowLabel in the ECMP hash. Hosts then repath connections by changing the FlowLabel on their packets. Linux already supports this behavior for IPv6, building on the longstanding notion of host rerouting on transport failures, e.g., TCP timeouts. We completed this support by handling outages encountered by acknowledgement packets.

PRR repairs failed paths at RTT timescales and without added overhead. It is applicable to all reliable transports, including multipath ones (§2.5). It has a small implementation, and is incrementally deployable across hosts and switches.

For the last several years, PRR has run 24x7 on Google networks worldwide to protect all TCP and Pony Express [31] (an OS-bypass datacenter transport) traffic. Recently, we have extended it to cover Google Cloud traffic. The transition was lengthy due to the vendor lead times for IPv6 FlowLabel hashing, plus kernel and switch upgrades. But the availability improvements have been well worthwhile: fleetwide monitoring of our backbones over a 6-month study found that PRR reduced the cumulative region-pair outage time (sum of outage time between all pairs of regions in the network) over TCP with application-level recovery by 63–84%.

This work makes two contributions. We describe the nature of outages in a large network, using case studies to highlight long outages that undermine availability. And we present data on the effectiveness of PRR for our fleet as a whole, plus learnings from production experience. We hope our work will encourage others to leverage the IPv6 FlowLabel with an architecture in which routing provides hosts with many diverse paths, and transports shift flows to improve reliability.

This work does not raise ethical issues.

2 DESIGN

We describe PRR, starting with our network architecture, and then how outage detection and repathing work in it.

2.1 Multipath Architecture

PRR is intended for networks in which routing provides multiple paths between each pair of hosts, as is the case for most modern networks. For example, in Fig 1 we see an abstract WAN in which a pair of hosts use four paths at once out of many more possibilities. When there is a fault, some paths will lose connectivity but others may continue to work. In the figure, one path fails due to a switch failure, and another due to a link failure. But two paths do not fail despite these faults.

Multipath provides an opportunity for hosts to repair the fault for users. To do so, we need hosts to be able to control path selection for their connections. We shifted our IPv6 network architecture to use the FlowLabel [2] to accomplish this goal.

In the original IPv4 Internet, hosts were intentionally oblivious to the path taken by end-to-end flows. This separation between end hosts and routing was suited to a network that scaled by making each link run faster and thus had relatively few IP-level paths between hosts. Over the past two decades, networks have increasingly scaled capacity by adding many links in parallel, resulting in numerous additional viable paths. Flows must be spread across these paths to use the capacity well, and so multipathing with ECMP [24, 48] has become a core part of networks. The same paths that spread flows to use capacity for performance also increase diversity for reliability.

The IPv6 architecture allows hosts to directly manage the multiplicity of paths by using the FlowLabel to tell switches which set of packets needed to take the same path, without knowing the specifics of the path [34]. This arrangement lets transports run flows over individual paths while letting the IP network scale via parallel links. However, IPv6 was not widely deployed until around 2015, and this usage did not catch on. In the interim, ECMP used transport headers for multipathing, e.g., each switch hashes IP addresses and TCP/UDP ports of each packet to pseudo-randomly select one of the available next-hops. This approach eases the scaling problem, but limits each TCP connection to a fixed network path.

The adoption of IPv6 allows us to use the FlowLabel as intended. At switches, we include the FlowLabel in the ECMP hash, a capability which is broadly available across vendors[1]. As a pragmatic step, we extend ECMP to use the FlowLabel along with the usual 4-tuple. The key benefit is to let hosts change paths without changing transport identifiers. In Fig 1, one connection may be shifted across the four paths simply by changing the FlowLabel.

2.2 High-Level Algorithm

PRR builds on the concept of host rerouting in response to transport issues. An instance of PRR runs for each connection at a host to protect the forward path to the remote host. This is necessary because connections use different paths due to routing and ECMP, so one instance of PRR cannot learn working paths from another.

An outage for a connection occurs when its network path loses IP connectivity. If the connection is active during an outage, PRR will detect an outage event via the transport after a short time, as described in §2.3. The outage may be a forward, reverse, or bidirectional path failure. Unidirectional failures are quite common since routes between hosts are often asymmetric, due to traffic engineering onto non-shortest paths [23].

PRR triggers repathing intended to recover connectivity in response to the outage event, as described in §2.4. If the new path successfully recovers connectivity, the outage ends for the connection (even though the fault may persist). If not, PRR will detect a subsequent outage event after an interval, and again trigger repathing. It will continue in this manner until either recovery is successful, the outage ends for exogenous reasons (e.g. repair of the fault by the control plane), or the connection is terminated.

A fundamental ambiguity in this process is that a host alone cannot distinguish forward from reverse or bidirectional path failure. As a result, outage events may cause PRR to perform spurious repathing, which can slow recovery by causing a forward path failure where none existed. Fortunately, this does not affect correctness because outage events will continue to trigger repathing until both forward and reverse paths recover.

2.3 TCP Outage Detection

PRR can be implemented for all reliable transports. We describe how it works with TCP as an important example. Since network outages are uncommon, PRR must have acceptable overhead when there is no outage. This means PRR must be very lightweight in terms of host state, processing and messages. Our approach is to detect outages as part of normal TCP processing.

Data Path. There are many different choices for how to infer an outage, from packet losses to consecutive timeouts. For established connections, PRR takes each TCP RTO (Retransmission TimeOut) [35] in the Google network as an outage event. This method provides a signal that recurs at exponential backoff intervals while connectivity is sufficiently impaired that the connection is unable to make forward progress. It is possible that an RTO is spurious or indicates a remote host failure, but repathing is harmless in these situations (as it is either very likely to succeed or won't help because the fault is not network related).

[1]Support was limited when we began but has increased steadily.

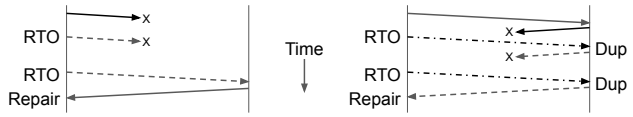


Figure 2: Example recovery of a Unidirectional Forward (left) and Reverse (right) fault. Non-solid lines indicate a changed FlowLabel.

ACK Path. Interestingly, RTOs are not sufficient for detecting reverse path outages. Failure of the ACK path will not cause an RTO for the ACK sender because ACKs are not themselves reliably acknowledged. Consider request/response traffic. The response will not be sent until the request has been fully received, which will not happen if the pure ACKs for a long request are lost.

We use the reception of duplicate data beginning with the second occurrence as a signal that the ACK path has failed. A single duplicate is often due to a spurious retransmission or use of Tail Loss Probes (TLP) [10], whereas a second duplicate is highly likely to indicate ACK loss.

Control Path. Finally, we detect outages in an analogous way when a new connection is being established. We use SYN timeouts in the client to server direction, and reception of SYN retransmissions in the server to client direction.

Performance. The performance of PRR depends on how quickly these sequences are driven by RTOs. Outside Google, a reasonable heuristic for the first RTO on established connections is RTO = 3RTT, with a minimum of 200ms. Inside Google, we use the default Linux TCP RTO formula [36] but reduce the lower-bound of RTTVAR and the maximum delayed ACK time to 5ms and 4ms from the default 200ms and 40ms, respectively [8]. Thus a reasonable heuristic is RTO = RTT + 5ms. It yields RTOs as low as single digit ms for metropolitan areas, tens of ms within a continent, and hundreds of ms for longer paths. These lower RTOs speed PRR by 3–40X over the outside heuristic. For new connections, a typical first SYN timeout is 1s, at the upper end of RTOs. This implies that connection establishment during outages will take significantly longer than repairing existing connections.

Examples. Recovery for request/response traffic with a unidirectional path fault is shown in Fig 2. For simplicity the figures focus on the initial transmission and later ones involving repathing, while omitting other packets present for fast recovery and TLP, etc. Each non-solid line indicates a changed FlowLabel.

In the forward case, the behavior is simple: each RTO triggers retransmission with repathing (dashed line). This continues until a working forward path is found. Recovery is then direct since the reverse path works. The number of repaths (here two) is probabilistic. The expected value depends on the fraction of working paths.

The reverse case differs in two respects. The RTOs cause spurious repathing (dot-dash line) since the forward path was already working. However, it is not harmful because only the reverse direction is faulty. Duplicate detection at the remote host (after TLP which is not shown) causes reverse repathing until a working path is found. The recovery time is the same as the forward case despite these differences.

Bidirectional faults are more involved because the repair behavior depends on whether the connection initially failed on the

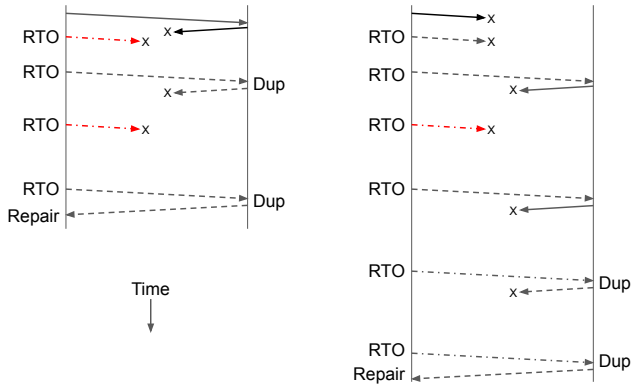


Figure 3: Example recovery of a Bidirectional fault when only the Reverse path (left) or both directions (right) initially failed. Non-solid lines indicate a changed FlowLabel. Red dash-dot lines indicate harmful repathing.

forward path, reverse path, or in both directions. Unlike the case of a unidirectional fault, we may have harmful spurious repathing and delayed reverse repathing.

If the connection initially experiences a forward path failure but (by chance) no reverse failure, then recovery proceeds as for the unidirectional case. When the forward repair succeeds, it is the first time the packet reaches the receiver. Thus the receiver will not repath. It will use its working reverse path to complete recovery.

Conversely, if the connection initially fails only on the reverse path (Fig 3 left) then we see different behavior. At the first RTO, spurious forward repathing occurs. Now it can be harmful, and is in this example (dot-dash red line). It causes a forward path loss that requires subsequent RTOs to repair. When the forward path works, duplicate reception causes reverse repathing. We must draw both a working forward and reverse path at the same time to recover.

The longest recovery occurs when the connection has failed in both directions (Fig 3 right). Reverse repathing is needed for recovery but delayed until after two repairs of the forward path. This is because the receiver first repaths on the second duplicate reception and the original transmission and TLP are lost. To complete recovery, we need a subsequent joint forward and reverse repair. As before, spurious forward repathing may slow this process.

2.4 Repathing with the FlowLabel

PRR repaths as a local action by using the FlowLabel. It does not rely on communication with other network components (e.g., SDN controllers or routing) during outages.

Random Repathing. For each outage event, PRR randomly changes the FlowLabel value used in the packets of the connection. This behavior has been the default in Linux via `txhash` since 2015, with our last upstream change (for repathing ACKs) landing in 2018. The OS takes this action without involving applications, which greatly simplifies deployment.

With a good ECMP hash function, different inputs are equivalent to a random draw of the available next-hops at each switch. If we define a path as the concatenation of choices at each switch, then paths more than a few switches long will change with very high probability. However, the key question is how often a new path will intersect the fault.

The fraction of failed paths for the outage must be measured empirically since it depends on the nature of the fault. It also varies for each prefix-pair and changes over time. Often the outage fraction is small, leading to rapid recovery. For example, with a 25% outage (in which a quarter of the paths fail) a single random draw will succeed 75% of the time. More generally, for an IP prefix-pair with a p% outage, the probability of a connection being in outage after N rerouting attempts falls as p^N.

Avoiding Cascades Repathing needs to avoid cascade failures, where shifting a large amount of traffic in response to a problem focuses load elsewhere and causes a subsequent problem. PRR is superior to routing in this respect: fast-reroute shifts many "live" connections in the same way at the same time, while PRR shifts traffic more gradually and smoothly.

The shift is gradual because each TCP connection independently reacts to the outage, which spreads reaction times out at RTO timescales. Each connection is quiescent when it moves, following an RTO, and will ramp its sending rate under congestion control.

The shift is smooth because random repathing loads working paths according to their routing weights. The expected load increase on each working path due to repathing in one RTO interval is bounded by the outage fraction. For example, it is 50% for a 50% outage: half the connections repath and half of them (or a quarter) land on the other half of paths that remain. This increase is at most 2X, and usually significantly lower, which is no worse than TCP slow-start [25] and comfortably within the adaptation range of congestion control. Moreover, PRR can only use policy-compliant paths which have acceptable properties such as latency, while there are no such guarantees for bypass routes.

A related concern is that repathing in response to an outage will leave traffic concentrated on a portion of the network after the outage has concluded. However, this does not seem to be the case in practice: routing updates spread traffic by randomizing the ECMP hash mapping, and connection churn also corrects imbalance.

2.5 Alternatives

Multipath Transports. A different approach is to use multipath transports such as MPTCP [6] or SRD [38] since connections that use several paths are more likely to survive an outage; these transports can also improve performance in the datacenter [33]. However, PRR may be applied to any transport to boost reliability, including multipath ones. For example, MPTCP can lose all paths by chance, and it is vulnerable during connection establishment since subflows are only added after a successful three-way handshake. PRR protects against these cases.

Moreover, many connections are lightly used, so the resource and complexity cost of maintaining multiple paths does not have a corresponding performance benefit. The prospect of migrating all datacenter usage to a new transport was also a consideration. Instead, we apply PRR to TCP and Pony Express to increase the reliability of our existing transports.

Application-Level Recovery. Applications can approximate PRR without IPv6 or the FlowLabel by reestablishing TCP connections

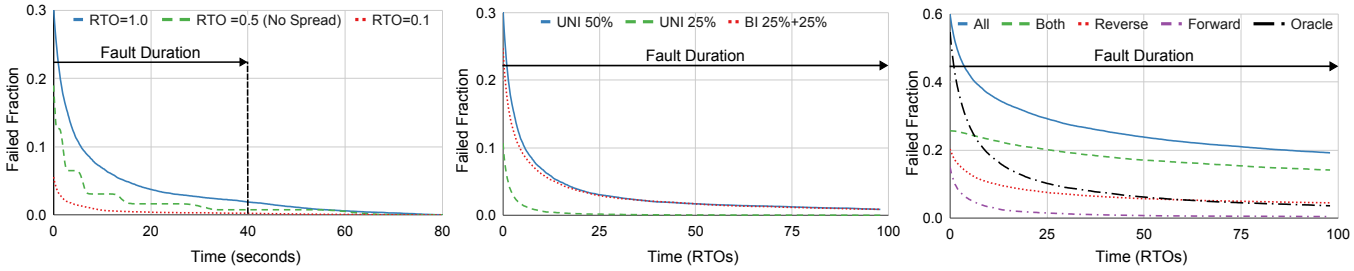


Figure 4: (a) Effect of RTO (b) Uni- and bi-directional repair curves (c) Breakdown of bidirectional repair

that have not made progress after a timeout[2]. We relied on this approach before PRR. However, using low RPC timeout values is much more expensive than PRR, as well as less performant. This is because establishing an RPC channel takes several RTTs and has computational overhead for security handshakes. Coverage is also more limited since it relies on application developers. Adding PRR to TCP covers all manner of applications, including control traffic such as BGP and OpenFlow, whether originating at switches or hosts, and achieves the ideal of avoiding application recovery.

PLB. Finally, PRR is closely related to Protective Load Balancing [32]. In our implementation, they are unified to use the same repathing mechanism but for different purposes. PLB repaths using congestion signals (from ECN and network queuing delay) to balance load. PRR repaths using connectivity signals (e.g., timeouts) to repair outages. These signals coexist without difficulty, but there is one point of interaction. PRR activates during an outage to move traffic to a new working path. Since outages reduce capacity, it is possible that PLB will then activate due to subsequent network congestion and repath back to a failed path. Therefore, we pause PLB after PRR activates to avoid oscillations and a longer recovery.

3 SIMULATION RESULTS

We use a simple model to predict how PRR reduces the fraction of failed connections. Our purpose is to provide the reader with a mental model for how PRR is expected to perform; we will see these effects in our case studies (§4.2).

We simulate repathing driven by TCP exponential backoff for an ensemble of 20K long-lived connections under various fault models. This workload represents the active probing that we use to measure connectivity. The fault starts at $t = 0$ and impacts each connection when it first sends at $t \geq 0$. We model black hole loss and ignore congestive loss. A connection is considered failed if a packet is not acknowledged within 2s. The repair behavior is strongly influenced by two factors that we consider in turn: the RTO (Retransmission TimeOut), and the outage fraction of failed paths.

RTO. The RTO depends on the connection RTT and state, and the TCP implementation. For our network the RTT ranges from O(1ms) in a metro, to O(10ms) in a continent, to O(100ms) globally. For new connections, the SYN RTO is commonly 1s. This variation has a corresponding effect on the speed of recovery.

Fig 4(a) shows the repair of a 50% outage (i.e., half the paths fail) in one direction for three different RTO scenarios. The middle line in the graph is the repair curve for connections having RTOs clustered around a median of 0.5s, with most mass from 0.45 to 0.55s. They are generated with a log-normal distribution, LogN(0, 0.06), scaled by the median RTO of 0.5s. The connections also have 1s of jitter in their start times. The aggregate behavior is a "step" pattern, where the failed fraction is reduced by 50% when the connections repath at each step. Note that the failed fraction starts at around 0.2, much lower than the 50% of connections that were initially black holed. This is because the black holed connections RTO and most recover before the 2s timeout.

While instructive, we only see this step pattern on homogeneous subsets of data. More typical are the smooth curves of the top and bottom lines due to connections repathing at different times. They have median RTOs of 1s and 100ms, respectively, and are generated with LogN(0, 0.6) scaled by the median RTO. This distribution spreads the RTOs to have a standard deviation that is 10X larger than before. The bottom line shows a much faster repair due to the lower median RTO and has become smooth due to the RTO spread. The smaller 100ms RTO makes a large difference. It both reduces the initial failed fraction and reaches successive RTOs more quickly. The top line, with initial RTOs around 1s, models the failure rate for new connections as well as long RTTs. It shows a correspondingly slower repair due to the larger RTO.

For both curves, the failed fraction of connections, f, falls polynomially over time. Suppose the outage fails a fraction paths p. After N RTOs, f is p^N below its starting value, which is exponentially lower. However, the RTOs are also exponentially-spaced in time, t, so we have $t \approx 2^N$ for the Nth RTO. Combining expressions, $f \approx p^{log_2(t)} = 1/(t)^K$, for $K = -log_p(2)$. Thus for $p = \frac{1}{2}$, the failure probability falls as $1/t$. For $p = \frac{1}{4}$, it falls as $1/t^2$.

A notable effect is that the fault (dashed line) ends at $t = 40$s, yet some connections still lack connectivity until $t = 80$s. That is, the failures visible to TCP can last longer than the IP-level outage. The reason is exponential backoff. It is not until the first retry after the fault has resolved that the connection will succeed in delivering a packet and recover. If the fault ends at $t = 40$s then some connections may see failures in the interval $[20, 40)$. These connections will increase their RTO and retry in the interval $[40, 80)$.

Outage Fraction. The severity of the fault has a similarly large impact on recovery time. Fig 4(b) shows repair for three different long-lived faults. We normalize time in units of median initial RTOs, spread RTOs as before, and use a timeout of twice the median RTO.

The top solid line is for a 50% outage in one direction. It corresponds to the 1s RTO curve from before. The bottom solid line shows the repair of a 25% outage in one direction. It has the same

[2]Linux fails TCP connections after ~15 mins by default. Application timeouts are shorter.

effects, but starts from a lower failed fraction and falls more quickly. Now, each RTO repairs 75% of the remaining connections.

The dashed line shows the repair of a bidirectional outage in which 25% of paths fail in each direction. For each connection, the forward and reverse paths fail independently to model asymmetric routing. This curve is similar to the 50% unidirectional outage, even though it might be expected to recover more quickly since the probability of picking a working forward and reverse path is $\frac{9}{16}$, which is larger than $\frac{1}{2}$. The reason is that the bidirectional outage has three components that repair at different rates, as we see next.

In Fig 4(c), we break the repair of a long-lived 50% forward and 50% reverse outage (solid line) into its components (dashed lines) as described in §2.3. This outage is demanding, with 75% of the round-trip paths having failed, so the tail falls by only one quarter at each RTO. Connections that initially failed in one direction only, either forward or reverse, are repaired most quickly. Connections that initially failed in both directions are repaired slowly due to spurious repathing and the delayed onset of reverse repathing. To see the cost of these effects, the Oracle line (dotted) shows the how the failed fraction improves without them.

Summary. We conclude that for established connections with small RTOs, PRR will repair >95% of connections within seconds for faults that black hole up to half the paths. This repair is fast enough that the black holes are typically not noticed. Larger RTOs and new connections will require tens of seconds for the same level of repair, and show up as a small service interruption. PRR cannot avoid a noticeable service interruption for the combination of large RTOs and faults that black hole the vast majority of paths, though it will still drive recovery over time.

4 PRODUCTION RESULTS

PRR runs in Google networks 24x7 and fleetwide for TCP and Pony Express traffic. We present a measurement study to show how it is able to maintain high network availability for users, beginning with case studies of outages and ending with fleet impact.

Our study observes real outages at global scale across the entire production network. It covers two backbones, B2 and B4 [23, 26], that use widely differing technologies (from MPLS to SDN) and so have different fault and repair behaviors. Further, the results we derive are consistent with service experience to the best of our knowledge. One limitation of our study is that we are unable to present data on service experience. We report results for long-lived probing flows instead. Still, our measurements are comprehensive, with literally trillions of probes.

4.1 Measuring loss

We monitor loss in our network using active probing. We send probes between datacenter clusters using multiple flows, defined as source/destination IP and ports. Flows take different paths due to ECMP. Each flow sends ~120 probes per minute. Each pair of clusters is probed by at least 200 flows. This arrangement lets us look at loss over time and loss over paths, with high resolution in each dimension. We also aggregate measurements to pairs of network regions, where each region is roughly a metropolitan area and contains multiple clusters.

We use three types of probes to observe loss at different network layers. First, UDP probes measure packet loss at the IP level. We refer to these probes as L3. They let us monitor the connectivity of the underlying network, which highlights faults and routing recovery, but not how services experience the network.

To measure application performance before PRR, we use empty Stubby RPCs as probes; Stubby is an internal version of the open source gRPC [18] remote procedure call framework. We refer to these probes as L7. They benefit from TCP reliability and RPC timeouts, which reestablish TCP connections that are not making progress. An L7 probe is considered lost if the RPC does not complete within 2s. Stubby reestablishes TCP connections after 20s to match the gRPC default timeout.

Finally, to measure application performance with PRR, we issue the L7 probes with PRR enabled, which we refer to as L7/PRR. These probes benefit from PRR repathing as well as TCP reliability and RPC timeouts. Note that the network may be in outage while applications are not, due to PRR, TCP and RPC repair mechanisms. Thus comparing the three sets of probes lets us understand the benefits of PRR relative to applications without it and the underlying network.

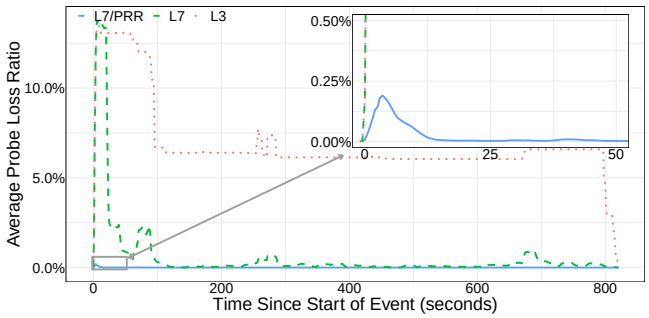
Our fleet summaries use probe data for 6 months to the middle of 2023 between all region-pairs in our core network, and on both backbones. Since the functionality to disable PRR is not present on all probe machines, we note that there are slightly fewer L7 probes (29%) than L7/PRR (37%) and L3 probes (33%), but we have no reason to believe the results are not representative.

4.2 Case Studies

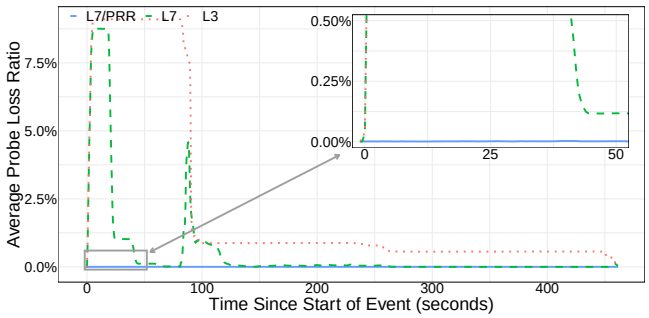
We begin with case studies of how PRR behaves during a diverse set of significant outages. Most outages are brief or small outages. The long and large outages we use for case studies are exceptional. They are worthy of study because they are highly disruptive to users, unless repaired by PRR or other methods.

Case Study 1: Complex B4 Outage. The first outage is the longest we consider, lasting 14 mins. It impacted region-pairs connected by the B4 backbone. We use it to highlight multiple levels of repair operating at different timescales. It was caused by a rare dual power failure that took down one rack of switches in a B4 supernode [23] and disconnected the remainder of the supernode from an SDN controller. It was prolonged by a repair workflow that was blocked by unrelated maintenance. Long outages have diverse causes and typically complex behaviors. In this case, a single power failure would not have led to an outage, and a successful repair workflow would have led to a much shorter outage, but in this unlucky case three events happened together.

The probe loss during the outage is shown in Fig 5. The top graph shows the loss versus time for L3, L7 and L7/PRR probes over impacted inter-continental region-pairs over B4. The bottom graph shows the same for intra-continental pairs. We separate them since the behaviors are qualitatively different. One datapoint covers 0.5s, so the graphs show a detailed view of how the fault degraded connectivity over time. Each datapoint is averaged over many thousands of flows sending in the same interval, which exercised many thousands of paths.

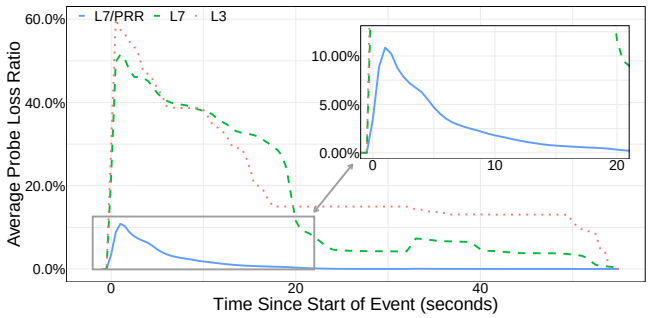


(a) Inter-continental probe loss

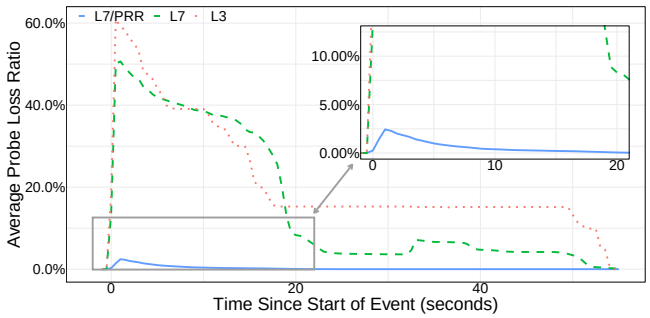


(b) Intra-continental probe loss

Figure 5: Probe loss during a complex B4 outage.



(a) Inter-continental probe loss



(b) Intra-continental probe loss

Figure 6: Probe loss during an optical link failure on B4.

Consider first the L3 line, which shows the IP-level connectivity. Since the SDN control plane was disconnected by the fault, it could not program fixes to drive a fast repair process. Around 100s, global routing systems intervened to reroute traffic not originating from or destined for the outage neighborhood. This action reduced the severity of the outage but did not fix it completely. After more than 10 mins, the drain workflow removed the faulty portion of the network from service to complete the repair.

The loss rate stayed below 13% throughout the outage because only one B4 supernode was affected by the fault so most flows transited healthy supernodes. However, the outage was more disruptive than the loss rate may suggest because the failure was bimodal, as is typical for non-congestive outages: all flows taking the faulty supernode saw 100% loss, while all flows taking the healthy supernodes saw normal, low loss. For customers using some of the faulty paths, the network was effectively broken (without PRR) because it would stall applications even though most paths still functioned properly.

The L7 line shows the outage recovery behavior prior to the development of PRR. The L7 loss rate started out the same as L3, but dropped greatly after 20s, after which it decayed slowly, with occasional spikes. The L7 improvement is due to the RPC layer, i.e., application-level recovery, which opened a new connection after 20s without progress. These new connections with different port numbers avoided the outage by taking different network paths due to ECMP. Most of the new paths worked by chance because the L3 loss rate tells us that on average only 13% of paths initially failed. This is similar to the repathing done by PRR except that (1) by using the FlowLabel, PRR can repath without reestablishing the TCP connection; and (2) PRR operates at RTT timescales, which are much shorter than the 20s RPC timeout.

Some TCP flows required multiple attempts to find working paths, so there was a tail as connectivity was restored. The subsequent spikes arose when routing updates changed paths, altering the ECMP mapping and causing some working connections to become black holed. TCP retransmissions were of little help in this process, since a connection either worked or lost all of its packets at a given time.

Finally, the L7/PRR line shows the outage recovery using the same RPC probes as the L7 case but with PRR enabled. The loss rate was greatly reduced, to the point of being visible only in the inset panel. The repair was roughly 100X more rapid than the L7 case, especially for the intra-continental case due to its shorter RTT. It achieved the desired result: most customers were unaware that there was an outage because the connectivity interruption is brief and does not trigger application recovery mechanisms.

This case study highlights outage characteristics that we observe more broadly. Many outages black hole a fraction of paths between a region-pair while leaving many other paths working at the same time. And some outages are not repaired by fast reroute so they have long durations that are disruptive for users without quick recovery. Outages with both characteristics provide an opportunity for PRR to raise network availability.

Case Study 2: Optical failure. Next we consider an optical link failure that resulted in partial capacity loss for the B4 backbone. Fig 6 shows the probe loss over time for inter- and intra-continental paths during the outage. In this case, L3 loss was around 60% when

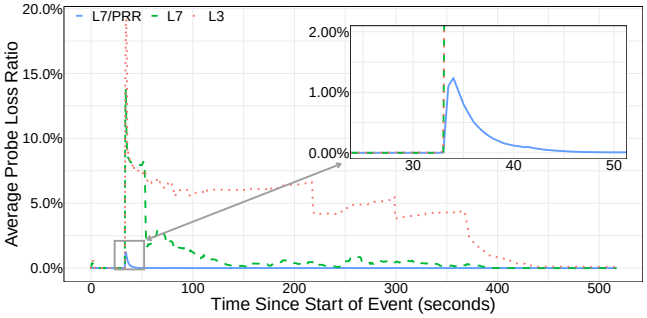


Figure 7: Inter-continent probe loss during a device failure on B2. (No intra-continent probe loss was observed.)

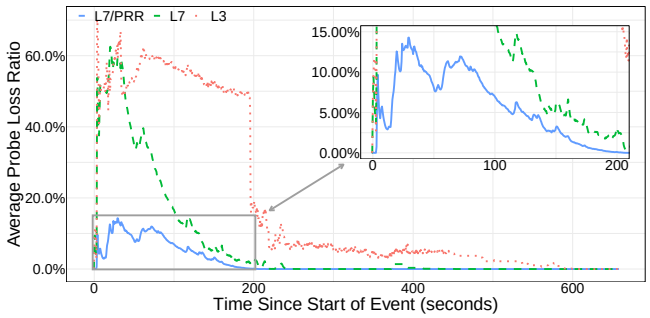


Figure 8: Intra-continental probe loss for a regional fiber cut in B2. (The inter-continental graph is omitted as similar.)

the event began, indicating that most paths had failed but there was still working capacity.

Immediately after the outage began, fast routing repair mechanisms acted to reduce the L3 loss to ~40% within 5s. Further improvements gradually reduced the L3 loss to ~20% by around 20s from the start of the event. The cause of this sluggish repair was congestion on bypass links due to the loss of a large fraction of outgoing capacity, and SDN programming delays due to the need to update many routes at once. Finally, the outage was resolved after 60 seconds when unresponsive data plane elements were avoided using traffic engineering.

The L7 line starts out slightly lower then the L3 one because L7 probes have a timeout of 2s, during which time routing was able to repair some paths. We see that TCP was unable to mitigate probe loss during the first 20s since retransmissions do not help more than routing recovery. In fact, after around 10s, L7 loss exceeded L3 loss because the detection of working paths was delayed by exponential backoff. Around 20s, RPC channel reestablishment roughly halves the loss rate for the remainder of the outage for both intra- and inter-continental paths. All these effects are consistent with our simulation results (§3).

In contrast, PRR lowered peak probe loss and quickly resolved the outage. For intra-continental paths, L7/PRR reduced the peak probe loss to 2.4% and had completely mitigated the loss by 20s into the outage. L7/PRR similarly performed well for inter-continental paths where probe loss peaked at around 11%, which is over 5X less than the peak L3 probe loss. This outage illustrated how the path RTT affects PRR. Consistent with simulation (§3), intra-continental paths that have lower RTTs observed a lower peak and faster resolution than inter-continental paths. In both cases, PRR greatly reduced probe loss beyond L7.

Case Study 3: Line card issues on a single device. The next outage involved a single device in our B2 backbone (Fig 7). During the outage, the device had two line-cards malfunction, which caused probe loss for some inter-continental paths. Due to the nature of the malfunction, routing did not respond. The outage was eventually mitigated when an automated procedure drained load from the device and took it out of service.

While the cause of this outage is different than the others, we see similar results: PRR was able to greatly reduce loss. In this case, the

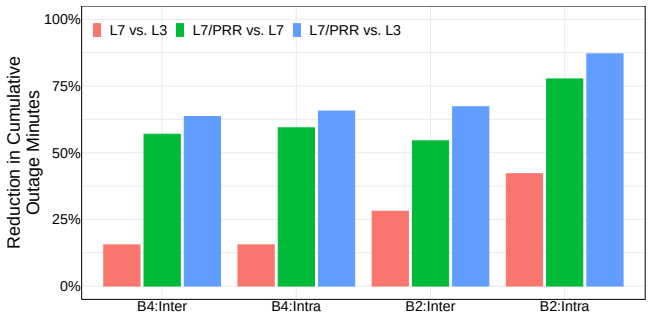


Figure 9: Reduction in outage minutes for B2 and B4 intra- and inter-continental paths.

peak probe loss seen by L3 was 19%. L7/PRR reduced this peak loss over 15X to 1.2% and, as with the prior outage, quickly lowered the loss level to near zero after 20 seconds. Conversely, the L7 probe loss has a large peak of 14% and persists for significantly longer than L7/PRR.

Case Study 4: Regional fiber cut. Finally, we present an outage that challenged PRR (Fig 8). In this outage, a fiber cut caused a significant loss of capacity. The average L3 probe loss peaked at 70% and remained around 50% or higher for 3 mins. This severe fault impacted many paths. Fast reroute did not mitigate it because the bypass paths were overloaded due to the large capacity reduction. After ~3 mins, global routing moved traffic away from the outage, lowering the loss rate and alleviating congestion on the bypass paths.

L7/PRR reduced the peak loss to 14%, a 5X improvement. It was much more effective than L7, which reduced peak loss to 65%, but was not able to fully repair this outage because of the large fraction of path loss. This path loss was exacerbated by routing updates during the event: PRR moved connections to working paths only for some of the connections to shift back to failed paths when the ECMP mapping was changed. As a result, we see a pattern in which L7/PRR loss fell over time but was interrupted with a series of spikes.

4.3 Aggregate Improvements

Case studies provide a detailed view of how PRR repairs individual outages. To quantify how PRR raises overall network availability, we aggregate measurements for all outages across all region-pairs in the Google network for the 6-month study period. The vast majority of the total outage time is comprised of brief or small outages. Our results show that PRR is effective for these outages, as well as long and large outages.

Outage Minutes Our goal is to raise availability, which is defined as $MTBF/(MTBF + MTTR)$, where $MTBF$ is the Mean Time Between Failures and $MTTR$ is the Mean Time to Repair. This formula is equivalent to 1 minus the fraction of outage time. Since we are unable to report absolute availability measures due to confidentiality, we report relative reductions in outage time across L3, L7, and L7/PRR layers. These relative reductions translate directly to availability gains. For instance, a 90% reduction in outage time is equivalent to adding one "nine" to availability, e.g., hypothetically improving from 99% to 99.9%

We measure outage time for each of L3, L7 and L7/PRR in minutes derived from flow-level probe loss. Specifically, we compute the probe loss rate of each flow over each minute. If a flow has more than 5% loss, such that it exceeds the low, acceptable loss of normal conditions, then we mark it as lossy. If a 1-minute interval between a pair of network regions has more than 5% of lossy flow, such that it is not an isolated flow issue, then it is an outage minute for that region-pair. We further trim the minute to 10s intervals having probe loss to avoid counting a whole minute for outages that start or end within the minute.

In Fig 9, we show the percent of total outage minutes that were repaired for L7/PRR probes relative to L3 probes. We give results for both B2 and B4 backbones, and broken out by intra- and inter-continental paths. Similarly, we compare L7 (without PRR) with L3 and L7/PRR with L7 to understand where the gains come from. The results are computed across many thousands of region-pairs and include hundreds of outage events.

PRR reduces outage minutes by 64-87% over L3. PRR combined with transport and application layer recovery mechanisms is very effective at shortening IP network outages for applications. We see large reductions in outage minutes when using L7/PRR for both backbone networks. The reductions range from 64% for inter-continental paths on B4, to 87% for intra-continental paths on B2. Note that, unlike for individual outages, we do not see a consistent pattern between inter- and inter-continental results across outages. This is because PRR effectiveness depends on topological factors and not only the RTT.

PRR reduces outage minutes by 54-78% over L7. PRR is able to repair most of the outage minutes that are not repaired by the TCP and application-level recovery of L7 probes. This result confirms that most of the L7/PRR improvement over L3 is not due to the RPC layer and TCP retransmissions; L7 reduces the cumulative outage minutes by only 15–42% relative to L3. We believe that a large portion of this gain is coming from RPC reconnects, since TCP retransmission is ineffective for black holes.

PRR performs well over time. As a check, we also look at how PRR behavior varies over time. Fig 10 shows the Generalized Additive Model (GAM) smoothing [43] of the fraction of daily outage minutes repaired. We see some variation over time, reflecting the varying nature of outages, while PRR consistently delivers large reductions in outage minutes throughout the study period.

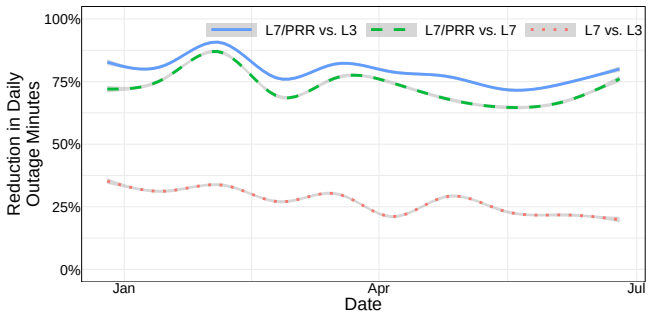


Figure 10: Fraction of outage minutes reduced over time.

4.4 Effectiveness for Region-Pairs

Outages affect multiple region-pairs, each of which may see a different repair behavior. We next look at how the benefit of PRR is distributed across region-pairs in our network. Fig 11 shows the Complementary Cumulative Distribution Function (CCDF) over region-pairs of the fraction of outage minutes repaired between layers. This graph covers all region-pairs in the fleet over our entire study period. Points higher and further to the right are better outcomes as they mean a larger fraction of region-pairs repaired a greater fraction of outage minutes.

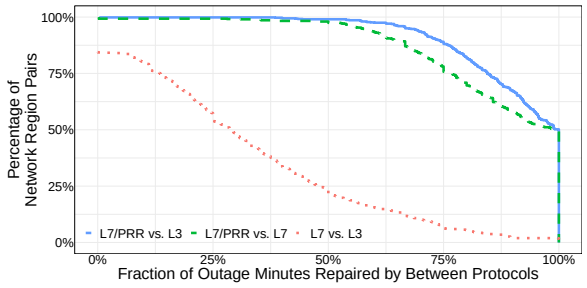
PRR performs well for a variety of paths. We see that the vast majority of region-pairs see a large benefit from L7/PRR over L3 on both backbones. It is able to repair 100% of outage minutes for 50% and 16% of B2 intra- and inter-continental region-pairs, respectively. PRR performance is more varied for B4 where outage minutes are decreased by half for 63% and 77% of intra- and inter-continental region-pairs, respectively.

PRR improves significantly over L7. As expected, L7/PRR provides much greater benefit than L7. The lines showing PRR gain are quite similar whether they are relative to L3 or L7 and show a reduction in outage minutes for nearly all region-pairs. (The exceptions tend to be region-pairs with very few outage minutes for which L7/PRR dynamics for sampling were unlucky.) Conversely, L7 without PRR increases the number of outage minutes relative to L3 for 3-16% of region pairs. This counter-intuitive result is possible because TCP exponential backoff on failed paths tends to prolong outage times (until RPC timeouts are reached).

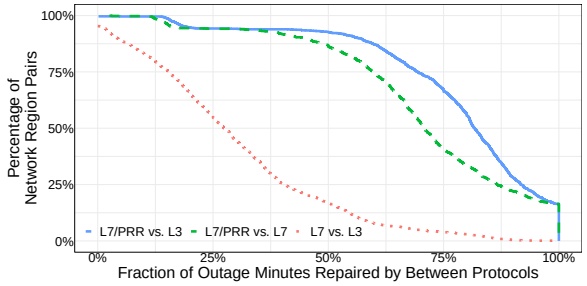
5 DISCUSSION

We briefly discuss some additional aspects of PRR.

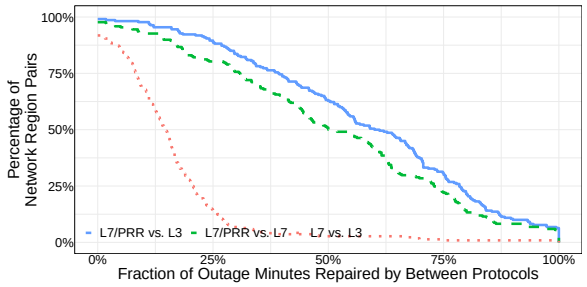
Other Transports. PRR can be applied to protect all transports, including multipath ones, since all reliable transports detect delivery failures. For example, we use PRR with Pony Express [31] OS-bypass traffic with minor differences from TCP. User-space UDP transports can implement repathing by using syscalls to alter the FlowLabel when they detect network problems. Even protocols



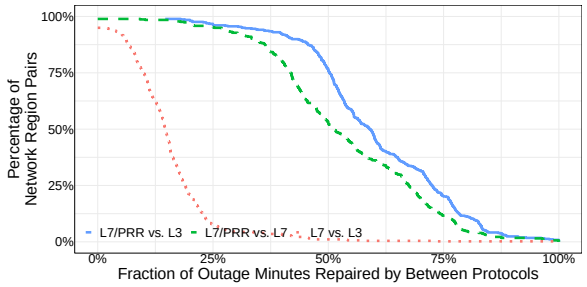
(a) intra-continental (B2)



(b) inter-continental (B2)



(c) intra-continental (B4)



(d) inter-continental (B4)

Figure 11: CCDF of improvement across region pairs.

such as DNS and SNMP can change the FlowLabel on retries to improve reliability.

Cloud & Encapsulation. PRR must be extended to protect IPv6 traffic from Cloud customers because virtualization affects ECMP. Google Cloud virtualization [12] uses PSP encryption [19], adding IP/UDP/PSP headers to the original VM packet as shown in Fig 12. In the network, switches use the outer headers for ECMP and ignore the VM packet headers. To enable the VM to repath via the FlowLabel, we hash the VM headers into the outer headers.

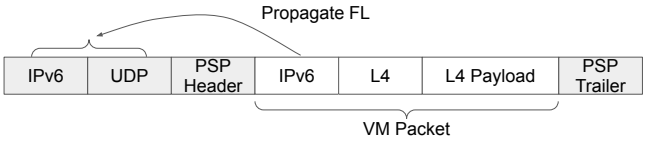


Figure 12: PSP Encapsulation

Now, when the guest OS has PRR and a TCP connection detects an outage and changes its FlowLabel, the encapsulation headers also change and hence ECMP causes the connection to be repathed. For different encapsulation formats, e.g., IPSEC, the details will vary, but the propagation approach is the same.

We also use encapsulation to enable PRR for Cloud IPv4 traffic. The gve [20] driver passes connection metadata to the hypervisor, which hashes it into the encapsulation headers. This technique works for other IPv4 networks as well: encapsulate with IPv4 to provide a layer of indirection, and propagate inner header entropy to an outer UDP header (since there is no FlowLabel).

Deployment. Backwards-compatibility and incremental deployment are highly desirable to protect existing deployments. PRR excels in this respect. Its rollout was lengthy due to vendor participation, upstream kernel changes, and switch upgrades, but otherwise straightforward. Hosts could be upgraded in any order to use the FlowLabel for outgoing TCP traffic. Switches could be concurrently upgraded in any order to ECMP hash on the FlowLabel; this change is harmless given the semantics of the FlowLabel.

It is not necessary for all switches to hash on the FlowLabel for PRR to work, only some switches upstream of the fault. Often, substantial protection is achieved by upgrading only a fraction of switches. This property has implications for the reliability of IPv6 traffic in the global Internet. Not only can each provider enable FlowLabel hashing to protect their own network, but upstream providers have some ability to work around downstream faults by changing the ingress into downstream networks.

6 RELATED WORK

Most work on improving network reliability focuses on network internals such as fast reroute [3, 4], backbones [23, 40, 47], or datacenters [44]. Fewer papers report on the causes of outages [17, 21, 41] or availability metrics suited to them, such as windowed-availability [22], which separates short from long outages.

Hosts have the potential to raise availability using the FlowLabel [5], but no large-scale deployment or evaluation has been presented to the best of our knowledge. Most host-side work focuses on multipath transports like MPTCP [6], SRD [38], and multipath QUIC [13] that send messages over a set of network paths. While they primarily aim to improve performance, these transports also increase availability, e.g., MPTCP may reroute data in one subflow to another upon RTO. However, they use only a small set of paths and may not protect the reliability of connection establishment, e.g., MPTCP adds paths only after a successful three-way handshake [5]. PRR can be added to multipath transports to increase reliability by exploring new paths until it finds working ones, and protecting connection establishment.

There is a large body of work on a related host behavior: multipathing for load balancing [1, 6, 14–16, 27–30, 32, 37, 39, 42, 45, 46]. Some of this work considers reliability, for example, CLOVE [28] uses ECMP to map out working paths. PRR shows this mapping is not necessary, as random path draws work well. Most of this work is not done in the context of IPv6 and does not consider the FlowLabel; we find it apt for multipathing. In our network, PRR and PLB [32] are implemented together, using repathing for both load balancing and rerouting around failures.

Finally, [7] argues that hosts should play a greater role in path selection, instead of routers reacting to failures. PRR is one realization of this argument.

7 CONCLUSION

PRR is deployed fleet-wide at Google, where it has protected the reliability of nearly all production traffic for several years. It is also available to our Cloud customers. PRR greatly shortens user-visible outages. In a 6-month study on two network backbones, it reduced the cumulative region-pair outage time over TCP with application-level recovery by 63–84%. This is the equivalent of adding 0.4–0.8 "nines" to availability. We now require that all our transports use PRR.

PRR (and its sister technique, PLB [32]) represent a shift in our network architecture. In the early Internet [11], the network instructed hosts how to behave, e.g., ICMP Source Quench. This did not work well, and in the modern Internet neither hosts nor routers instruct each other: hosts send packets, and routers decide how to handle them.

In our architecture, hosts instruct the network how to select paths for their traffic by using the IPv6 FlowLabel. This shift has come about because networks scale capacity by adding parallel links, which has greatly increased the diversity of paths. To make the most of this diversity, we rely on routing to provide hosts access to many paths, and hosts to shift traffic flows across paths to increase reliability and performance. We hope this architectural approach, enabled by the FlowLabel, will become widespread.

REFERENCES

[1] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 503–514. https://doi.org/10.1145/2619239.2626316

[2] Shane Amante, Jarno Rajahalme, Brian E. Carpenter, and Sheng Jiang. 2011. IPv6 Flow Label Specification. RFC 6437. (Nov. 2011). https://doi.org/10.17487/RFC6437

[3] Alia Atlas, George Swallow, and Ping Pan. 2005. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090. (May 2005). https://doi.org/10.17487/RFC4090

[4] Alia Atlas and Alex D. Zinin. 2008. Basic Specification for IP Fast Reroute: Loop-Free Alternates. RFC 5286. (Sept. 2008). https://doi.org/10.17487/RFC5286

[5] Alexander Azimov. 2020. Self-healing Network or The Magic of Flow Label. https://ripe82.ripe.net/presentations/20-azimov.ripe82.pdf. (2020).

[6] Olivier Bonaventure, Christoph Paasch, and Gregory Detal. 2017. Use Cases and Operational Experience with Multipath TCP. RFC 8041. (Jan. 2017). https://doi.org/10.17487/RFC8041

[7] Matthew Caesar, Martin Casado, Teemu Koponen, Jennifer Rexford, and Scott Shenker. 2010. Dynamic Route Recomputation Considered Harmful. *ACM SIGCOMM Computer Communication Review* 40, 2 (Apr. 2010), 66–71. https://doi.org/10.1145/1764873.1764885

[8] Neal Cardwell, Yuchung Cheng, and Eric Dumazet. 2016. TCP Options for Low Latency: Maximum ACK Delay and Microsecond Timestamps, IETF 97 tcpm. https://datatracker.ietf.org/meeting/97/materials/slides-97-tcpm-tcp-options-for-low-latency-00. (2016).

[9] Sid Chaudhuri, Gisli Hjalmtysson, and Jennifer Yates. 2000. Control of lightpaths in an optical network. In *Optical Internetworking Forum*.

[10] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. 2021. The RACK-TLP Loss Detection Algorithm for TCP. RFC 8985. (Feb. 2021). https://doi.org/10.17487/RFC8985

[11] David Clark. 1988. The Design Philosophy of the DARPA Internet Protocols. *SIGCOMM Comput. Commun. Rev.* 18, 4 (aug 1988), 106–114. https://doi.org/10.1145/52325.52336

[12] Mike Dalton, David Schultz, Ahsan Arefin, Alex Docauer, Anshuman Gupta, Brian Matthew Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, Jake Adriaens, Jesse L Alpert, Jing Ai, Jon Olson, Kevin P. DeCabooter, Marc Asher de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*. USENIX Association, Renton, WA, 373–387.

[13] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*.

[14] Advait Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. 2013. On the impact of packet spraying in data center networks. In *2013 Proceedings IEEE INFOCOM*. 2130–2138. https://doi.org/10.1109/INFCOM.2013.6567015

[15] Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, and Mohammad Alizadeh. 2016. Juggler: A Practical Reordering Resilient Network Stack for Datacenters. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 20, 16 pages. https://doi.org/10.1145/2901318.2901334

[16] Soudeh Ghorbani, Zibin Yang, P. Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro Load Balancing for Low-Latency Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 225–238. https://doi.org/10.1145/3098822.3098839

[17] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. *SIGCOMM Comput. Commun. Rev.* 41, 4 (aug 2011), 350–361.

[18] Google. 2015. gRPC Motivation and Design Principles (2015-09-08). https://grpc.io/blog/principles/. (2015).

[19] Google. 2022. PSP Architecture Specification (2022-11-17). https://github.com/google/psp/blob/main/doc/PSP_Arch_Spec.pdf. (2022).

[20] Google. 2022. Using Google Virtual NIC. https://cloud.google.com/compute/docs/networking/using-gvnic. (2022).

[21] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 58–72. https://doi.org/10.1145/2934872.2934891

[22] Tamás Hauer, Philipp Hoffmann, John Lunney, Dan Ardelean, and Amer Diwan. 2020. Meaningful Availability. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 545–557. https://www.usenix.org/conference/nsdi20/presentation/hauer

[23] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. 2018. B4 and after: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 74–87. https://doi.org/10.1145/3230543.3230545

[24] Christian Hopps and Dave Thaler. 2000. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991. (Nov. 2000). https://doi.org/10.17487/RFC2991

[25] Van Jacobson. 1988. Congestion Avoidance and Control. *SIGCOMM Comput. Commun. Rev.* 18, 4 (aug 1988), 314–329. https://doi.org/10.1145/52325.52356

[26] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-Deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.* 43, 4 (aug 2013), 3–14.

[27] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. 2014. FlowBender: Flow-Level Adaptive Routing for Improved Latency and Throughput in Datacenter Networks. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*. Association for Computing Machinery, New York, NY, USA, 149–160. https://doi.org/10.1145/2674005.2674985

[28] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. 2017. Clove: Congestion-Aware Load Balancing at the Virtual Edge. In *Proceedings of the 13th International Conference on Emerging*

Networking EXperiments and Technologies (CoNEXT '17). Association for Computing Machinery, New York, NY, USA, 323–335. https://doi.org/10.1145/3143361.3143401

[29] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. HULA: Scalable Load Balancing Using Programmable Data Planes. In *Proceedings of the Symposium on SDN Research (SOSR '16)*. Association for Computing Machinery, New York, NY, USA, Article 10, 12 pages. https://doi.org/10.1145/2890955.2890968

[30] Ming Li, Deepak Ganesan, and Prashant Shenoy. 2009. PRESTO: Feedback-Driven Data Management in Sensor Networks. *IEEE/ACM Transactions on Networking* 17, 4 (2009), 1256–1269. https://doi.org/10.1109/TNET.2008.2006818

[31] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Mike Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Mike Ryan, Erik Rubow, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: a Microkernel Approach to Host Networking. In *In ACM SIGOPS 27th Symposium on Operating Systems Principles*. New York, NY, USA.

[32] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: Congestion Signals Are Simple and Effective for Network Load Balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 207–218. https://doi.org/10.1145/3544216.3544226

[33] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. *SIGCOMM Comput. Commun. Rev.* 41, 4 (aug 2011), 266–277.

[34] Jarno Rajahalme, Alex Conta, Brian E. Carpenter, and Dr. Steve E Deering. 2004. IPv6 Flow Label Specification. RFC 3697. (March 2004). https://doi.org/10.17487/RFC3697

[35] Matt Sargent, Jerry Chu, Dr. Vern Paxson, and Mark Allman. 2011. Computing TCP's Retransmission Timer. RFC 6298. (June 2011). https://doi.org/10.17487/RFC6298

[36] Pasi Sarolahti and Alexey Kuznetsov. 2002. Congestion Control in Linux TCP. In *2002 USENIX Annual Technical Conference (USENIX ATC 02)*. USENIX Association, Monterey, CA. https://www.usenix.org/conference/2002-usenix-annual-technical-conference/congestion-control-linux-tcp

[37] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J. Freedman. 2013. Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*. Association for Computing Machinery, New York, NY, USA, 151–162. https://doi.org/10.1145/2535372.2535397

[38] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. 2020. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro* 40, 6 (2020), 67–73. https://doi.org/10.1109/MM.2020.3016891

[39] Shan Sinha, Srikanth Kandula, and Dina Katabi. 2004. Harnessing TCP's burstiness with flowlet switching. In *Proc. 3rd ACM Workshop on Hot Topics in Networks (Hotnets-III)*.

[40] Sucha Supittayapornpong, Barath Raghavan, and Ramesh Govindan. 2019. Towards Highly Available Clos-Based WAN Routers. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 424–440. https://doi.org/10.1145/3341302.3342086

[41] Daniel Turner, Kirill Levchenko, Alex C. Snoeren, and Stefan Savage. 2010. California Fault Lines: Understanding the Causes and Impact of Network Failures. *SIGCOMM Comput. Commun. Rev.* 40, 4 (aug 2010), 315–326. https://doi.org/10.1145/1851275.1851220

[42] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 407–420. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[43] Simon N Wood. 2017. *Generalized additive models: an introduction with R* (second ed.). Chapman and Hall/CRC, Boca Raton. https://doi.org/10.1201/9781315370279

[44] Dingming Wu, Yiting Xia, Xiaoye Steven Sun, Xin Sunny Huang, Simbarashe Dzinamarira, and T. S. Eugene Ng. 2018. Masking Failures from Application Performance in Data Center Networks with Shareable Backup. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 176–190. https://doi.org/10.1145/3230543.3230577

[45] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. 2012. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 139–150. https://doi.org/10.1145/2342356.2342390

[46] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient Datacenter Load Balancing in the Wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 253–266. https://doi.org/10.1145/3098822.3098841

[47] Zhizhen Zhong, Manya Ghobadi, Alaa Khaddaj, Jonathan Leach, Yiting Xia, and Ying Zhang. 2021. ARROW: Restoration-Aware Traffic Engineering. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 560–579. https://doi.org/10.1145/3452296.3472921

[48] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. 2014. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. Article No. 5. https://dl.acm.org/doi/10.1145/2592798.2592803

Regional IP Anycast: Deployments, Performance, and Potentials

Minyuan Zhou°*, Xiao Zhang★**, Shuai Hao†, Xiaowei Yang★, Jiaqi Zheng°,
Guihai Chen°, Wanchun Dou°
°State Key Laboratory for Novel Software Technology, Nanjing University, China,
★Duke University, †Old Dominion University

Abstract

Recent studies show that an end system's traffic may reach a distant anycast site within a global IP anycast system, resulting in high latency. To address this issue, some private and public CDNs have implemented regional IP anycast, a technique that involves dividing content-hosting sites into geographic regions, announcing a unique IP anycast prefix for each region, and utilizing DNS and IP-geolocation to direct clients to CDN sites in their corresponding geographic regions. In this work, we aim to understand how a regional anycast CDN partitions its sites and maps its customers' clients to its sites, and how a regional anycast CDN performs compared to its global anycast counterpart. We study the deployment strategies and the performance of two CDNs (Edgio and Imperva) that currently deploy regional IP anycast. We find that both Edgio and Imperva partition their sites and clients following continent or country borders. Furthermore, we compare the client latency distribution in Imperva's regional anycast CDN with its similar-scale DNS global anycast network, while accounting for and mitigating the relevant deployment differences between the two networks. We find that regional anycast can effectively alleviate the pathology in global IP anycast where BGP routes clients' traffic to distant CDN sites. However, DNS mapping inefficiencies, where DNS returns a suboptimal regional IP anycast address that does not cover a client's low-latency CDN sites, can harm regional anycast's performance. Finally, we show what performance benefits regional IP anycast can achieve with a latency-based region partition method using the Tangled testbed. When compared to global anycast, regional anycast significantly reduces the 90th percentile client latency by 58.7% to 78.6% for clients across different geographic areas.

CCS Concepts

• **Networks** → **Network measurement**; **Network performance analysis**; **Naming and addressing**; **Routing protocols**.

Keywords

Routing; IP Anycast; Regional Anycast

ACM SIGCOMM '23, September 10, 2023, New York, NY, USA

ACM 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3604846

ACM Reference Format:
Minyuan Zhou, Xiao Zhang, Shuai Hao, Xiaowei Yang, Jiaqi Zheng, Guihai Chen, Wanchun Dou. 2023. Regional IP Anycast: Deployments, Performance, and Potentials. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3603269.3604846

1 Introduction

IP anycast [50] refers to the routing practice where a network announces the same IP prefix from multiple geographically-distributed locations. It is widely used by distributed systems such as root Domain Name Service (DNS) servers and Content Distribution Networks (CDNs) to reduce client latency and balance load. Unlike conventional DNS-based redirection services [13], IP anycast can direct client traffic to nearby CDN sites without a separate load-balancing system [11]. Partly due to its simplicity, several large CDNs, including Cloudflare [18], Google Cloud CDN [17], and Microsoft Azure [51], have all adopted IP anycast.

The simplicity of IP anycast comes with a downside: it gives a CDN operator little control over which sites its clients' traffic reaches. The Border Gateway Protocol (BGP) [56] routes a client's traffic to a CDN site based on its policies and network topology dynamics. Since BGP is a policy-routing protocol, its best-path selection algorithm does not incorporate performance metrics directly associated with end-to-end path latency. As a result, BGP often routes a client to an anycast site that is geographically distant from the nearest anycast site [7, 8, 19, 23, 40, 42, 43, 58], leading to high client latency. We refer to this pathology as catchment inefficiency. Interactive applications such as gaming and web browsing demand low latency [2, 52]. In a competitive market, providing low-latency access to clients worldwide is crucial for CDN providers to gain competitive advantages.

Recently, several private and public CDNs adopted regional IP anycast as a promising approach to address some of the limitations of IP anycast [12, 31, 47]. A CDN that employs regional IP anycast partitions its sites into various geographical regions, such as continents or large countries, and announces a distinct IP anycast prefix from the sites in each region. When a client makes a DNS query to one of the CDN's customers, the CDN's DNS returns a regional IP anycast address based on the client's location. For clarity, we hereafter refer to the anycast configuration where a network announces the same IP prefix from multiple sites without regional partitions as global IP anycast.

Regional IP anycast retains the simplicity of IP anycast while providing CDN operators with a degree of control over the sites that a client's traffic can reach. However, this approach has not been

*Both authors contributed equally to this research.

thoroughly studied. Questions such as *how a regional IP anycast CDN is deployed* and *whether regional IP anycast can effectively address the catchment inefficiency problem of global IP anycast* are unanswered. Understanding these questions can provide valuable insights into optimizing the deployment and performance of large-scale IP anycast systems.

This work aims to answer the above questions. We first conduct an in-depth study on the deployment strategies and the performance of two global-scale regional anycast CDNs: Edgio (formerly Edgecast and acquired by Limelight in 2022) and Imperva (formerly Incapsula and now part of Imperva). According to a prior study [31] and our recent survey (§ 4.1), these two CDNs are among the top-15 largest CDNs that currently deploy regional IP anycast. Furthermore, we discover that Imperva's own authoritative DNS server system uses global IP anycast and its sites and network configurations overlap significantly with its regional anycast CDN. Therefore, we choose Imperva's authoritative DNS server system as its global anycast counterpart and compare their performance differences. We use RIPE Atlas [66], a globally distributed set of probes, to send DNS queries to the domains hosted by Edgio and Imperva. We then send traceroute queries from RIPE Atlas probes to the IP addresses the probes receive and infer the geographic locations of the CDN sites the probes' traffic reaches, which we refer to as the catchment sites. From these steps, we are able to infer the regional site partition and the DNS mapping strategies of the two CDNs (§ 4.4), as well as the anycast sites of Imperva's DNS server system.

We find that Edgio and Imperva use different regional partition strategies (§ 4.3). Edgio divides its customers' clients into three or four regions, while Imperva divides its customers' clients into six regions. The region boundaries in both CDNs largely follow country or continent borders. Most clients receive IP prefixes originating from the CDN sites in the same geographical areas from DNS, but sometimes DNS returns a remote regional IP address to a client. We reached out to both Edgio and Imperva to discuss our findings and one responded and confirmed parts of our findings.*

The performance study of regional IP anycast reveals both the advantages and limitations of regional IP anycast (§ 5). We find that regional IP anycast can effectively limit the worst-case catchment inefficiencies experienced by global IP anycast by directing clients to regional IPs. For instance, regional anycast reduces the 90th percentile client latency for Imperva in North America from 110 ms to 38 ms. However, compared to global IP anycast, regional IP anycast suffers DNS mapping sub-optimality. DNS may map a client to a sub-optimal regional IP that does not include the client's low-latency CDN sites, offsetting regional IP anycast's advantages of reducing catchment inefficiencies. For instance, DNS mapping inefficiencies increase the 90th percentile client latency for Imperva in Latin America from 93 ms to 102 ms.

Finally, we examine the performance benefits of regional anycast without encountering sub-optimal DNS mapping. We conduct this study using the Tangled testbed, an open-access anycast testbed that allows researchers to run customized anycast experiments (§ 6). We use a latency-based scheme to partition the Tangled testbed into regions and assign each RIPE Atlas probe to the region that includes its lowest-latency site. We then deploy both global IP anycast and regional IP anycast on the Tangled testbed. In this case, regional IP anycast can achieve lower client latency than global anycast in all geographical regions. This result highlights the performance potentials of regional IP anycast.

An inherent limitation of this work is that we measure client latency using RIPE Atlas, like many previous studies [39, 42, 49]. Using a different set of clients, one may observe different latency values. Despite this limitation, we believe this work makes the following general contributions:

- We study in detail the deployments and performance of two regional IP anycast CDNs and compare one CDN's performance with a comparable global IP anycast system. To the best of our knowledge, this work is the first extensive study of regional IP anycast CDNs.
- We validate the performance advantages of regional anycast experimentally and discover its drawbacks in certain circumstances. We show that regional IP anycast can effectively mitigate the worst-case catchment inefficiency problem experienced by global IP anycast by directing clients to regional IP anycast addresses, but DNS mapping sub-optimality may offset some of this effect.
- We experiment with a latency-based region partition and client mapping scheme that addresses DNS mapping inefficiencies using the Tangled testbed. We find that this method reduces the latency for RIPE Atlas probes in all geographic areas compared to global anycast. This experiment shows the performance potentials of regional IP anycast.

Ethical Considerations Active measurements such as issuing pings and BGP announcements can cause extra load on the Internet infrastructure. We mitigate these concerns by conducting our measurements at reasonably low rates (*i.e.*, only one round of ping or traceroute for each anycast IP address) with publicly accessible infrastructure. Our BGP announcements use only prefixes that Tangled controls and Tangled's AS number. The prefixes we use do not serve any clients. We only measured the CDN providers with ping and traceroute, and we did not retrieve webpages which could incur extra costs for their customers. This work raises no other ethical issues.

2 Background and Motivation

In this section, we discuss the catchment inefficiency problem of IP anycast, the challenges in addressing it, and our motivation to study regional IP anycast.

2.1 The Catchment Inefficiency Problem

IP anycast is a popular technique used by CDNs to direct client traffic to their sites [31]. With this technique, a CDN operator relies on the inter-domain routing protocol BGP to select the site a client reaches. However, being a policy-routing protocol, BGP often fails to route a client to a low-latency site. Figure 1 shows an example we observe in our measurements. The CDN we study (Imperva) has two involved sites: one connected to Level 3 in Ashburn, Virginia, and the other connected to SingTel in Singapore. When both sites announce the same global IP anycast prefix, the probe located in Washington D.C. reaches the Singapore site, as SigTel is the

*Due to confidentiality agreement, we cannot disclose which provider responded to our inquiries or what parts of our findings were confirmed.

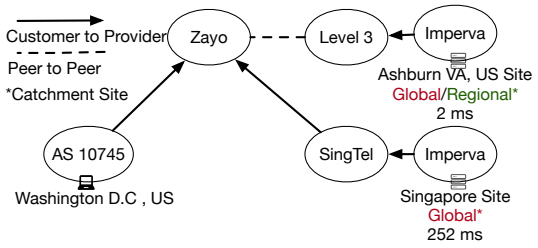


Figure 1: ***An example observed from Imperva's measurement results: with a global anycast configuration, the probe located in Washington D.C. reaches the CDN site in Singapore, while with regional anycast, the probe reaches the site in Ashburn, Virginia.***

customer of the probe's provider Zayo [28], while Level 3 is Zayo's peer. Under common BGP policies, ISPs prefer customer routes to peer routes. So the round trip time from this probe to the CDN is inflated by 250 ms. Furthermore, for routes with the same policy preference, BGP uses other metrics, such as AS path lengths, which are poorly correlated to performance, to select routes. Since a large AS may span multiple continents, routes with shorter (or the same) AS path lengths may have longer latencies than routes with longer (or the same) AS path lengths. A recent study [39] shows that for Microsoft CDN (a global IP anycast system), nearly 30% of users experience more than 30 ms latency inflation.

2.2 Challenges

There exist several proposals to improve client latency in a global anycast system. Ballani *et al.* [8] proposed to deploy anycast sites within a single provider. This proposal effectively limits BGP's policy routing, but it is sometimes necessary to connect an anycast system to many ISPs for scalability and robustness. Li *et al.* [42] proposed to introduce a new BGP attribute that encodes an anycast prefix's geographical origin. However, introducing changes to BGP is difficult in practice.

Alternatively, McQuistin *et al.* [49] proposed DailyCatch, a system that uses routine measurement to choose between a transit-provider-only and an all-peer configuration for an anycast system. This approach can effectively choose the better configuration between the two measured configurations, but can not optimize beyond that. Catchment inefficiencies can exist under either configuration. Zhang *et al.* [69] proposed AnyOpt, which uses pair-wise BGP experiments to choose an optimal site configuration for an anycast system among all possible site deployments, but pair-wise BGP experiments increase the operating overhead of CDN networks.

2.3 Regional IP Anycast

Regional IP anycast emerged as a promising approach to address the catchment inefficiency problem [12, 31, 47]. With regional IP anycast, a CDN divides its sites into multiple distinct geographic regions. It then assigns a distinct IP anycast prefix to each region. Sites in the same region will announce the same IP anycast prefix. We refer to such an IP prefix or address as a *regional IP prefix/address* or *regional IP* for short. The CDN then configures its DNS servers to assign a regional IP to a client based on the client's location. In the example shown in Figure 1, with the regional anycast configuration, the CDN announces different IP anycast prefixes in U.S. and Asia. The probe in the U.S. receives the U.S. IP prefix, and consequently, it reaches the Virginia site and enjoys a 2 ms RTT.

Unlike other proposals, regional IP anycast does not require changes to BGP, nor does it impose restrictions on how an anycast network connects to its providers. It eliminates the need for periodic BGP experiments and is complementary to DailyCatch, as it can mitigate catchment inefficiencies across various provider configurations.

2.4 Motivation

Despite its potential advantages, regional IP anycast is not well studied or widely deployed. In a blog article [47], LinkedIn described how they migrated their private CDN to a "prototype" regional anycast system and measured its client latency distribution. But their study is limited to LinkedIn's private CDN, which is primarily located in North and South America. Hao *et al.* [31] reported that two out of the top 20 CDNs (Edgio and Imperva) employ regional IP anycast without further performance analysis. Calder *et al.* [12] discussed the performance difference for Microsoft CDN when it employs a regional vs. a global anycast configuration.

This work aims to understand how regional anycast is deployed in practice and experimentally examine its performance benefits compared to global anycast. This study can provide new insights into how to design an anycast-based system that achieves low client latency worldwide. Additionally, if this study experimentally validates the performance benefits of regional anycast, it could motivate more CDNs to adopt regional anycast.

3 Measurement Infrastructure

We use two publicly available platforms: RIPE Atlas [66] and the Tangled testbed [9] to conduct our experiments. Our experiments were conducted in August 2022.

3.1 RIPE Atlas

RIPE Atlas is a measurement infrastructure that has more than 11,000 probes distributed around the world, each with the ability to execute pre-defined measurements periodically. Each probe's geographic location is publicly available. We leverage RIPE Atlas's user-defined measurement feature to send DNS queries, ping packets, and traceroute queries. We take the following steps to address the limitations of RIPE Atlas to suit our measurement purposes.

First, RIPE Atlas probes use user-reported geo-locations, which may contain errors, but we use the probes' built-in geocodes as ground truth to calculate the distance between a probe and its catchment site. To mitigate this issue and obtain stable measurement results, we discard the following probes: (1) probes with unreliable geocodes using the methods described in [29] and (2) probes that do not have a built-in stability tag (*e.g.*, "system-ipv4-stable-1d"). After this step, we retain 9,700+ out of 11,000+ RIPE Atlas probes.

Second, RIPE Atlas probes are unevenly distributed across different geographic areas and Autonomous Systems (ASes). An uneven distribution may lead to under- or over-estimation of performance in certain geographic areas or ASes. To address this limitation, we group the probes by <city, AS> pairs and present statistics based on probe groups as in [49]. We obtain the city code of a probe by mapping the probe to its closest airport within the same country and using the airport's International Air Transport Association (IATA)

code [34] as the probe's city code. We use the probe's built-in AS number to identify its AS. Then, we use the median value measured from each <city, AS> probe group to represent the performance of a client residing in the same city and AS. Without specific mention, all CDFs, percentage, and percentile values we present here are computed based on probe groups, rather than individual probes. We obtain 6100+ unique probe groups.

Finally, RIPE Atlas has much more probes in Europe and North America than in other continents. Such bias may lead to results that either overestimate or underestimate the performance of regional IP anycast in different regions. To overcome this limitation, we present the performance results of the probes in different geographic areas separately. We categorize the probes into four geographic areas based on their density:

- **EMEA**: Europe, Middle East, and Africa. This area has 3,859 unique probe groups and 6,917 unique probes.
- **NA**: North America, excluding countries in Central America. This area has 1,154 unique probe groups and 1,716 unique probes.
- **LatAm**: South America and countries in Central America. This area has 141 unique probe groups and 177 unique probes.
- **APAC**: the rest of the globe. This area has 613 unique probe groups and 950 unique probes.

We note that this area definition is based on the location of a RIPE Atlas probe and is independent of the CDN region partition schemes we soon discuss.

3.2 The Tangled Testbed

Tangled is a worldwide open-access IP anycast testbed. It has 12 sites distributed around the world. We use Tangled to evaluate a latency-based regional anycast scheme and to compare global anycast with latency-based regional anycast (§ 6.2). We also considered the PEERING testbed [59], but opted for the Tangled testbed because the PEERING testbed has no site in Asia and the Pacific area. We list the distribution of Tangled sites by geographic area in Table 1.

4 Deployments

In this section, we dissect the deployments of two regional IP anycast CDNs: Edgio and Imperva. We describe how we conduct measurements to answer the following questions:

- How do these regional IP anycast CDNs assign their clients to regional IP addresses?
- How do these regional IP anycast CDNs partition their sites and announce regional IP prefixes?

4.1 Identifying Regional IP Anycast CDNs

First, we identify the set of public CDNs that deploy regional IP anycast. To do so, we acquire the top apex domain list from Tranco [41] in April 2022. An apex domain is a two-level domain [32], *e.g.*, `example.com`. We then use the CDNFinder's API [53] to identify the CDN providers of each domain. Specifically, we provide CDNFinder with the www-prefixed hostname (referred to as the website hereafter) of each apex domain, such as `www.example.com`. CDNFinder determines the CDN providers used by a website by analyzing the response header of each resource on its landing page. Since each resource identifier corresponds to a hostname, we can tally the number of hostnames served by a CDN provider. We choose the top-15 CDN providers ranked by the number of hostnames they serve. The top-15 CDN providers cover 65.7% of all Tranco's top-10k domains. By manually examining their official technical articles or configuration documents (see Appendix A), we identify that Edgio and Imperva are the only two CDNs that deploy regional anycast among the top-15 CDN providers, consistent with a prior study [31]. Therefore, we focus our study on these two CDNs.

4.2 Customers of Regional IP Anycast CDNs

Second, we select the representative customers of a regional IP anycast CDN. A CDN provider may negotiate different service packages with its customers and use different system configurations to implement different service packages. As this work does not aim to unveil various service packages of a CDN, we select and measure representative CDN customers to understand how the CDNs enable regional IP anycast for their customers in commercial platforms.

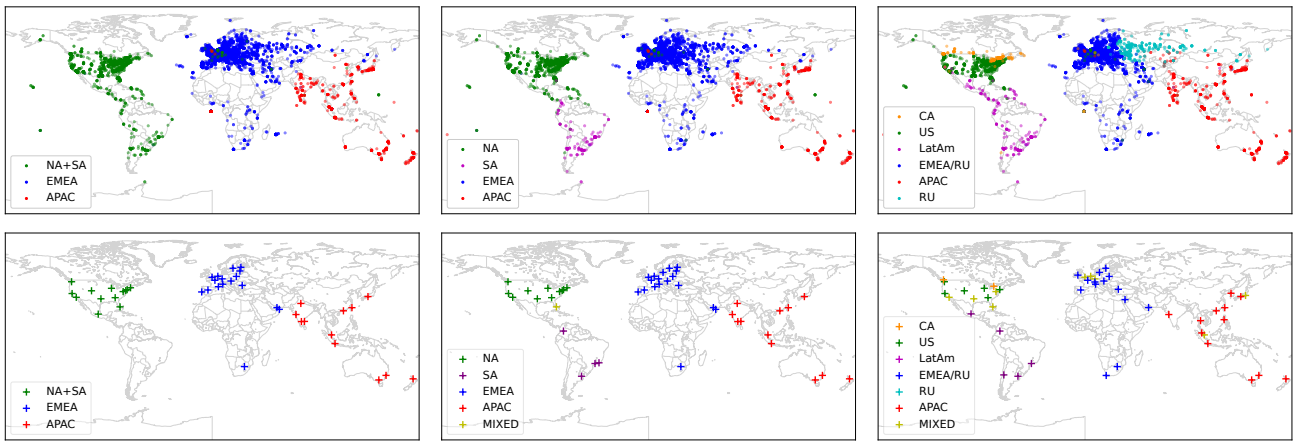
For this step, we first resolve all hostnames uncovered by CDNFinder to use Edgio or Imperva as their CDN providers to IP addresses. The results from CDNFinder in the previous step show that 2.98% of the top-10K websites are using Edgio or Imperva, in which Edgio serves 209 websites and Imperva serves 89 websites. We further extract 187 (96 and 91, respectively) distinct hostnames that point to Edgio or Imperva from these websites. To emulate a worldwide clientele, we compile a list of /24 client IP prefixes that cover the IP address span of the entire RIPE Atlas. We then use Google DNS [6] with the EDNS Client Subnet Extension (ECS) [20] to resolve all hostnames, an approach used in [10, 38].

We discover the number of unique IP prefixes each hostname resolves to and filter those hostnames that are not served by Edgio or Imperva's regional IP anycast networks. We record the IP address(es) (the `A` record) in each DNS response we receive. Out of Edgio's hostnames, 52.1% (50 out of 96) resolve to three distinct IP addresses each when accessed by the emulated worldwide clientele. We refer to this set of hostnames as *Edgio-3*. And 35.4% (34 out of 96) Edgio's hostnames resolve to four distinct IP addresses. We refer to this set of hostnames as *Edgio-4*. In contrast, the majority of Imperva's hostnames resolve to the same number of IP addresses. Specifically, 85.7% (78 out of 91) Imperva's hostnames resolve to six distinct IP addresses each. We refer to this set of hostnames as *Imperva-6*.

For the remaining hostnames that CDNFinder identifies as using Edgio or Imperva, they either resolve to a single IP address or multiple IP addresses matching the number of Edgio's or Imperva's published CDN sites, or IP addresses associated with other CDNs. We conclude that these hostnames are not (solely) served by Edgio's or Imperva's regional IP anycast CDN and exclude them from this study.

4.3 Client Partitions

To characterize which IP address(es) a client in a geographic region receives, we use RIPE Atlas probes to resolve a hostname that belongs to a customer of Edgio's or Imperva's regional IP anycast CDN. We then cluster RIPE Atlas probes based on the IP addresses they receive. For each hostname, we group the probes that receive the same IP address together. We run the experiments for all hostnames in the three sets: Edgio-3, Edgio-4, and Imperva-6. We find



(a) *Edgio-3 (www.straitstimes.com)* **(b)** *Edgio-4 (www.asus.com)* **(c)** *Imperva-6 (www.stamps.com)*

Figure 2: ***The first row shows the regional IP a RIPE Atlas probe receives in different regional anycast CDNs; the second row shows the CDN sites that announce a regional IP anycast prefix. Probes and CDN sites are color-coded by regional IPs. A site that announces more than one regional IP is shown in yellow with the legend "MIXED". Figures a & b show that Edgio divides its sites into four regions, but only returns three regional IPs to the clients of its Edgio-3 customers. Figure c shows Imperva divides its clients into six regions, but it has no sites in one of the regions (Russia).***

that the clustering results for each hostname in the same set remain the same.

To visualize client partitions, we pick one representative hostname from each hostname set that illustrates stable and consistent deployments while remaining top-ranked in Tranco's list. These hostnames are `www.straitstimes.com`, `www.asus.com`, and `www.stamps.com`*, which belong to the Edgio-3, Edgio-4, and Imperva-6 sets, respectively. The first row in Figure 2 shows how the probes that receive the same regional IP addresses for the three hostnames are distributed globally. We depict the probes that receive the same regional IP with the same color.

For both Edgio hostnames and Imperva hostnames, the client partition appears to happen at the continent or large-country level. For example, for Edgio-3 hostnames, the probes in North America and South America receive the same regional IP, and the probes in Europe, Africa, and Middle East receive the same regional IP. For Imperva hostnames, the probes in Russia receive different regional IPs from those in Europe; and the probes in the U.S. and Canada are also separated from each other, each receiving a distinct IP.

We use the country code of each RIPE Atlas probe to evaluate whether the probes in the same country always receive the same regional IPs for the same hostname and find that the majority of countries receive only one regional IP. For the probes from all 172 countries, 81.7%, 84.7% and 79.3% of the countries only receive one regional IP for the representative Edgio-3, Edgio-4, and Imperva-6 hostnames, respectively. For countries that receive two or more regional IPs, we find two cases. In the first case, the IP addresses of the probes are geolocated to different countries. For instance, the probes whose IPs belong to international transit providers are often geolocated to their home countries, not the countries they reside in. Second, the countries are either at the border of two regions or across two different continents. For instance, 10 out of 547 probes in Russia receive the EMEA regional IPs in Imperva-6.

Takeaways: Both Edgio and Imperva employ regional anycast CDNs that predominantly map clients to regional IPs based on geographic locations, such as continents or countries where the clients are located.

4.4 CDN Site Partitions

Next, we aim to understand how Edgio and Imperva partition their hosting sites into different regional IP anycast networks. To do so, we need to locate the CDN sites that announce a regional IP anycast prefix. We describe the site-mapping process as follows.

First, we obtain the locations of Edgio and Imperva's CDN sites. Both Edgio and Imperva publish their Points of Presence (PoPs) on their websites [24, 36]. We aggregate those locations at the city level and combine multiple PoPs in the same city as one site. We use these published site locations at the city level as the ground-truth locations of their sites.

Secondly, in order to determine the location of the CDN site announcing a regional IP, we perform traceroutes from each RIPE Atlas probe to the regional IP received by the probe from DNS. We then geolocate the IP address of the penultimate hop (referred to as p-hop hereafter) using the traceroute output. We use the ground-truth CDN site location to map each p-hop to its closest CDN site, as in [49]. This step also reveals the location of the catchment site of a probe, which we use to compute the distance between a probe and its catchment site in § 5.

IP geolocation is a task performed by many previous studies [5, 29, 42, 44, 45, 49, 64, 65]. This step involves substantial work, but it is not our main contribution. We summarize the work here and leave the detailed description in Appendix B. We first use the geo-hints in the reverse DNS (rDNS) name of a p-hop's IP to infer its location [44, 45]. If the geo-hints are unavailable, we use the location of a RIPE Atlas probe whose RTT is within 1.5 ms to the p-hop [29] to infer its location. The threshold is chosen according

*At the time when the measurement was carried out, `www.stamps.com` was still being hosted by Imperva.

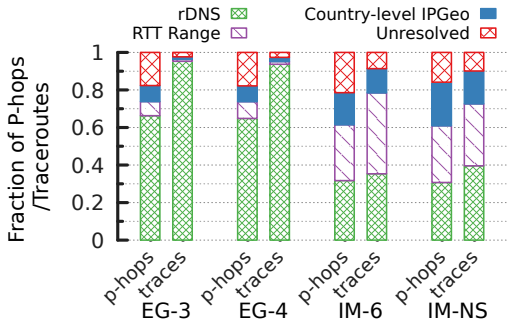


Figure 3: ***The p-hops bars illustrate the proportion of p-hops successfully IP-geolocated by a technique, as well as the fraction of unresolved p-hops for each network. The traces bars indicate the fraction of traceroutes with successfully IP-geolocated p-hops by a technique, as well as the fraction of traceroutes with unresolved p-hops. Abbreviations used: EG for Edgio, IM for Imperva, and IM-NS for Imperva's DNS server network.***

	EG-3	EG-4	EG-Pub	IM-6	IM-NS	IM-Pub	Tangled
APAC	14	15	19	16	17	17	2
EMEA	15	16	26	15	15	15	5
NA	13	12	24	12	12	12	3
LatAm	1	4	10	5	5	6	2
Total	43	47	79	48	49	50	12

Table 1: The number of sites in each geographic area of different networks *(EG/IM-Pub: Edgio and Imperva's Published sites).*

to the typical size of a metropolitan area, as the speed-of-light latency in fiber is roughly 100 km per 1 ms RTT. We refer to this technique as RTT Range. Third, if we cannot resolve a p-hop's geolocation after the previous two steps, we use the country information from three IP-geolocation databases (MaxMind [48], ipinfo [37], and EdgeScape [67]) to infer the location of the p-hop. If all IP-geolocation databases return the same country location for the p-hop, and the CDN provider only lists one site in the country, we use the listed site location as the p-hop's location. We refer to this technique as country-level IPGeo.

The site-mapping process described above requires sending traceroutes from RIPE Atlas probes and manual labor to examine uncommon geo-hints. Due to time and measurement traffic limitations, we are unable to map the CDN sites of all hostnames served by Edgio and Imperva. Instead, we mapped the CDN site partitions for four hostnames from each of the Edgio-3, Edgio-4, and Imperva-6 hostname sets, respectively. For each set, we include three randomly chosen hostnames and the representative hostnames used in § 4.3 and found that the site partitions for those hostnames are identical. In addition, we measure the client latency distribution for twelve additional Edgio-3, Edgio-4 and Imperva-6 hostnames, respectively, and show in Appendix C that the performance benefits of regional anycast to these additional Edgio and Imperva hostnames are similar. Therefore, we assume that the two CDNs we study either do not vary the site partition strategies among the hostnames in each hostname set we identify, or if they vary the site partition strategies for different hostnames, the variation does not significantly degrade the performance benefits of their regional anycast CDNs. Therefore, for an in-depth study, without specific mention, we present results for the representative hostnames only.

Figure 3 summarizes the fraction of p-hops we successfully geolocate for the representative Edgio-3, Edgio-4, and Imperva-6 hostnames by each technique. We also show the fraction of traceroutes whose p-hops are successfully IP-geolocated by each technique. As we can see, we are able to resolve the majority of the p-hops observed in the traceroute output.

Table 1 shows the CDN sites we uncover after mapping p-hops to their CDN sites. We uncover 48 out of 50 Imperva's published sites, and all 48 sites overlap with Imperva's DNS authoritative server sites. Edgio publishes 79 sites, while we only uncover 43 sites for the Edgio-3 hostname and 47 sites for the Edgio-4 hostname. Furthermore, we find that Edgio-3's CDN sites only overlap with 33 of its authoritative DNS server sites, while Edgio-4's CDN sites overlap with 37 of its DNS server sites. This result suggests that Edgio uses separate networks for its regional anycast CDN sites and DNS sites. Additionally, it employs distinct network configurations for other network services (§ 4.2) beyond the regional anycast CDN services we are studying. We spoke with one of the CDN providers and validated parts of our CDN and DNS site inferences.

CDN Site Partitions After the site-enumeration step, we are able to depict the location of a CDN site that announces a regional IP prefix. Figure 2 shows the result. We represent a CDN site with a colored cross symbol. The color corresponds to the regional IP it announces. We use the same color-coding scheme as in client partitions. If a site announces more than one regional IPs, we color the site yellow.

We make the following observations. First, in both Edgio and Imperva, client and site partitions are mostly consistent. Edgio partitions its sites into four regions, while Imperva into six regions. Clients in the same geographical regions receive the regional IPs originating from the CDN sites in the same regions except in two cases. For the Edgio-3 hostname, the probes in South America receive North America's regional IP. This finding shows that different customers receive different types of services from a CDN. Some customers' content may not be hosted by Edgio's South America sites. So clients will not be directed to those sites. For Imperva, although most probes in Russia receive distinct regional IPs as shown in Figure 2, these IPs are announced by three sites in Europe (Amsterdam, Frankfurt, and London), which also announce the EMEA regional IPs. We do not observe any Imperva sites in Russia.

Second, most sites only announce one regional IP prefix, but there exist sites that announce multiple regional IP addresses. We refer to this behavior as cross-region announcement. A closer examination of Figure 2b and Figure 2c show that the sites which announce multiple regional IPs usually locate near the border of two different regions. Cross-region announcements can help shorten client latency without adding additional sites. For instance, Edgio-4's "mixed" site in Florida, U.S. can serve both clients in North America and in South America. However, we later discover that cross-region announcements can lead to inefficient catchments, increasing the RTTs of some probes (§ 5.2).

Do CDN sites change the regional IP prefixes they announce over time? We enumerated the CDN sites that announce the regional IP

Condition	CDN	Local DNS				Authoritative DNS			
		APAC	EMEA	NA	LatAm	APAC	EMEA	NA	LatAm
$\Delta RTT < 5ms$	Edgio-3	94.2%	96.7%	98.6%	99.1%	94.2%	98.7%	98.8%	99.1%
	Edgio-4	91.0%	97.3%	97.4%	92.9%	94.8%	98.8%	98.2%	94.7%
	Imperva-6	86.3%	78.3%	88.0%	85.6%	87.1%	78.0%	89.3%	86.5%
✓Region, $\Delta RTT \geq 5ms$	Edgio-3	3.5%	1.3%	0.2%	0.0%	4.0%	0.8%	0.1%	0.0%
	Edgio-4	3.6%	0.7%	0.7%	3.5%	3.4%	0.7%	0.6%	3.5%
	Imperva-6	10.5%	19.4%	8.2%	11.7%	10.8%	21.0%	9.0%	12.6%
×Region, $\Delta RTT \geq 5ms$	Edgio-3	2.4%	2.0%	1.2%	0.9%	1.8%	0.5%	1.1%	0.9%
	Edgio-4	5.4%	2.0%	2.0%	3.5%	1.8%	0.4%	1.2%	1.8%
	Imperva-6	3.2%	2.3%	3.8%	2.7%	2.1%	1.0%	1.7%	0.9%

Table 2: We tabulate three types of DNS mapping results by percentages of probes: ✓/×Region indicates whether a probe receives a regional IP intended for its geographic location or vice versa; ΔRTT is the difference between a probe's RTT to the regional IP returned by DNS and the lowest one among its RTTs to all regional IPs. When ΔRTT exceeds 5 ms, we consider DNS mapping inefficient.

prefixes for three hostnames in Edgio-3, three hostnames in Edgio-4, and three hostnames in Imperva-6 weekly for two months. We find that for the same hostname, the sites that announce their regional IP prefixes in this two-month period remain the same.

Takeaways: Edgio and Imperva's site and client regional partitions are largely consistent, but there exist regions in which clients are assigned to regional IPs originating from CDN sites in different geographic regions.

4.5 Reachability of Regional IP addresses

Are regional IP prefixes globally reachable? We are interested in understanding whether a CDN restricts the BGP announcements of a regional IP anycast prefix to a geographic area. If it does, a client outside a geographic area cannot reach the regional IP prefix announced from that area. To do so, we send ping packets from RIPE Atlas probes to the regional IP addresses they do not receive from DNS. We run this experiment for each representative Edgio-3, Edgio-4, and Imperva-6 hostname. The results show that all probes can reach the regional IP addresses DNS returns to the probes in other regions. Global reachability provides robustness to regional anycast: even if DNS returns a regional IP unintended for a client's geographic area, the client can still reach the CDN site announcing the unintended regional IP.

5 Performance

In this section, we study the performance of Edgio's and Imperva's regional IP anycast CDNs. We aim to answer the following questions:

- How effectively does DNS map a client to the lowest-latency or a close-to-lowest-latency regional IP?
- What are the performance benefits of regional IP anycast compared to global IP anycast?

We use two metrics for this study: network latency and geographic distance.

Network latency or latency We measure a probe's round trip time (RTT) to its catchment site to quantify the client latency distribution achieved by an anycast system. The lower the latency, the better the performance.

Geographic distance We also measure the geographic distance between a probe and a CDN site as in previous work [22, 39, 42]. Since we have inferred the location of a probe's catchment site in § 4.4, we can compute the geographic distance between a probe and its catchment CDN site.

5.1 DNS Mapping Efficiency

To study DNS mapping efficiency, we measure how often DNS returns the lowest- or a close-to-lowest-latency regional IP to a client. We consider any regional IP with less than 5 ms RTT difference to a probe's lowest-latency regional IP as close-to-lowest regional IP. We instruct RIPE Atlas probes to send DNS queries to the representative hostnames served by Edgio and Imperva and record the returned IP addresses. Then we instruct each probe to ping all regional IPs associated with a hostname. Because DNS mapping results depend on whether a local resolver implements EDNS Client Subnet Extension [20], we run these experiments with two different DNS configurations:

Local DNS (LDNS) we configure each RIPE Atlas probe to use its local DNS resolver to send DNS queries when resolving a hostname.

Authoritative DNS (ADNS) For comparison, we configure each RIPE Atlas probe to send DNS queries directly to the authoritative name servers of a CDN, which are responsible for resolving customer domains to CDN's IPs. By doing this, the authoritative name servers of the CDN can determine the IP addresses they return based on the IP addresses of the querying clients.

We divide the experimental results into three groups and tabulate the results in Table 2. In the first group, DNS effectively returns a regional IP with less than 5 ms RTT difference to a client's lowest-latency regional IP ($\Delta RTT < 5\ ms$). We consider 5 ms a reasonable threshold to differentiate the performance of two CDN sites. We consider DNS mapping efficient in this case. In the second and third groups, DNS returns a regional IP whose RTT exceeds a client's minimum RTT among all regional IPs by 5+ ms.

We sub-divide the cases where a client receives a regional IP with an RTT more than 5 ms longer than the lowest-latency regional IP based on the causes of DNS mapping inefficiencies. According to the study in § 4.3, regional anycast CDNs map a client to a regional IP based on its geographic location; clients residing in the same geographic regions often receive the same regional IPs. If DNS maps a client to a regional IP outside its geographic area, we refer to this case as incorrect region mapping (×Region), which is likely due to IP geolocation errors. If DNS maps a client to a regional IP intended for clients in its geographic area (✓Region), but the RTT to this regional IP exceeds a client's RTT to the lowest-latency regional

IP by more than 5 ms, we consider this case as sub-optimal region mapping.

From Table 2, we can see that for Edgio's two representative hostnames, DNS maps more than 90% of the probes in all regions to regional IPs within 5 ms difference to their lowest-RTT regional IPs. Both incorrect region mapping and sub-optimal region mapping contribute to DNS inefficiencies. DNS mapping inefficiencies are more dominant in APAC and LatAm regions.

Imperva-6's DNS mapping is less efficient than Edgio's. The majority of the DNS inefficiencies are caused by inefficient regional mappings as shown in the (✓Region, $\Delta RTT \geq 5ms$) row in Table 2, due to Imperva-6's six-region partition scheme. Imperva-6 partitions the probes and sites in NA into two regions: Canada and the U.S. Around 10% probes in Canada and the U.S. are located near the Canada and U.S. border. Some of these probes have shorter RTTs to regional IPs in the other country. As a result, DNS maps only 88.0% of probes in NA to their low-latency regional IPs.

Similarly, Imperva-6 partitions Russia into a separate region. Probes in Russia receive distinct regional IPs which originate from three sites in Europe (Amsterdam, Frankfurt, and London). In this case, some probes in Russia have shorter RTT to EMEA (excluding Russia) regional IPs than to its own regional IPs and vice versa. For example, a probe in Russia reaches the site in Amsterdam, but its lowest-latency regional IP originates from a site in Copenhagen, Denmark. Its latency to the EMEA regional IP is 30 ms less than the latency to the Russian regional IP. As a result, for the EMEA region, only 78.3% of the probes receive regional IPs within 5 ms to their lowest-latency regional IPs in the Imperva-6 CDN.

Takeaways: This study reveals that regional anycast experiences DNS mapping inefficiencies, which causes 0%-21% of the probes in different regions to experience 5+ ms increased latency. Both incorrect regional mapping and DNS mapping based on a rigid regional partition can lead to sub-optimal performance.

5.2 Client Latency

In this section, we measure the performance of Edgio's and Imperva's regional anycast CDNs using the metrics we describe above: network latency and geographic distance. To measure client latency, we instruct a RIPE Atlas probe to ping the regional IP address it receives from DNS and record the RTT for each of the representative hostname of Edgio-3, Edgio-4, and Imperva-6 customers. In the meantime, we also plot the distance between a probe and the regional anycast site it reaches, as the speed-of-light latency lower-bounds the network latency. Figure 4 (a) and (b) show the cumulative distributions of the RTTs and the distance values of RIPE Atlas probes to Edgio-3, Edgio-4, and Imperva-6 regional anycast CDNs. We combine the measurement results for Edgio-3 and Edgio-4 for comparison.

We highlight two observations. First, the latency and distance values for probes in LatAm improve significantly in Edgio-4 compared to Edgio-3. The 80th percentile client latency decreases from 132 ms to 76 ms. Recall that Edgio maps the probes in South America to sites in North America in the Edgio-3 configuration. This result shows that mapping clients to the regional IPs of nearby CDN sites improves client latencies.

Percentile	Imperva-6 (Imperva-NS)			
	APAC	EMEA	NA	LatAm
80-th	38 (38)	31 (31)	25 (35)	68 (57)
90-th	63 (59)	45 (53)	38 (110)	102 (93)
95-th	98 (87)	67 (165)	54 (221)	120 (101)

Table 3: The tail latency comparison between Imperva-6 and its DNS global anycast network (Imperva-NS). RTTs are in the unit of millisecond. Green indicates a 5+ ms latency reduction in Imperva-6 and red indicates a 5+ ms increase.

Second, for both Edgio and Imperva, the 98th-percentile client latency in the NA and EMEA regions is less than 100 ms, the human interaction application threshold [52]. In the APAC region, however, more than 6.7% of Edgio-4 probe groups and 7.8% of Imperva-6 probe groups experience RTTs exceeding 100 ms. Similarly, in the LatAm region, more than 15.9% probe groups of Edgio-4 and 10.2% probe groups of Imperva-6 experience 100+ ms RTTs.

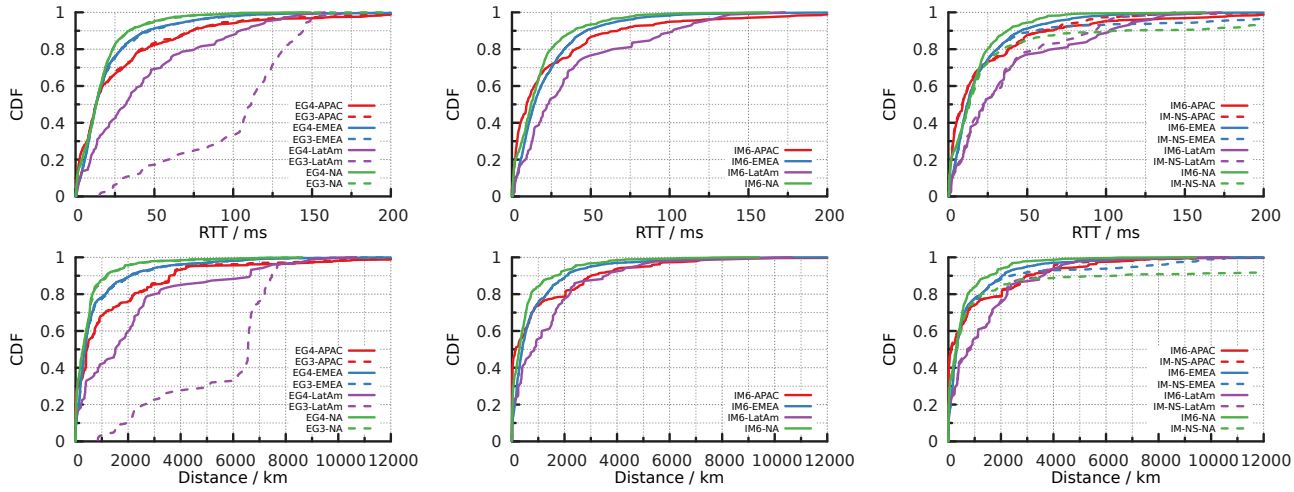
We investigate the reasons behind the occurrence of 100+ ms RTTs in regional anycast, using Imperva-6 as the specific case study. We find that a total of 148 probe groups of Imperva-6 experience 100+ ms RTTs. We categorize them into two sets: the first set comprises probe groups that have less than 100 ms RTTs to other alternative regional IPs. The second set consists of probe groups whose RTTs to all regional IPs exceed 100 ms. Among the probe groups in the first set, 48.0% of them receive the correct regional IPs from DNS, which are intended for their respective regions; the alternative regional IPs with less than 100 ms latencies lie outside their geographic regions. This result indicates that DNS's limitation to map a client to a regional IP based on the client's geographic location causes the 100+ ms latencies for those probe groups. For the remaining 52.0% of the probe groups in the first set, DNS returns regional IPs that do not correspond to their intended geographic areas, suggesting that IP geo-location errors are the underlying cause of the 100+ ms latencies. For probe groups in the second set, we have identified two factors that contribute to the 100+ ms latencies: cross-region announcements and poor intra-region connectivity. For example, Imperva has a site in California, US that announces its regional IPs for the APAC region. One probe group in China reaches this site and experiences 100+ ms latencies. In another example, a group of probes in Argentina reach their catchment site in Brazil via Italy, as there is no available network path between the probe group and the catchment site within the LatAm region.

5.3 Regional vs. Global Anycast

Next, we aim to study the performance impact of regional anycast when compared to global anycast.

A Comparable Global Anycast Counterpart Ideally, we aim to analyze the performance of a regional anycast CDN in comparison to the performance of the same CDN when configured to employ global anycast. This means announcing a global IP anycast prefix from all CDN sites, maintaining the same set of peers and policy configurations used for regional anycast. Such a study can unambiguously reveal the benefits or drawbacks of regional IP anycast.

Lacking access to a real-world CDN, we cannot conduct such a study. Instead, we use the DNS global anycast network of a regional anycast CDN to emulate its comparable global anycast counterpart. A common practice among CDN providers is to deploy their



(a) *Edgio-3 vs. Edgio-4* **(b)** *Imperva-6* **(c)** *Imperva-6 vs. Imperva-NS with same config*

Figure 4: ***The CDFs of client latency (first row) and the geographical distance to a client's catchment site (second row) of: (a) Edgio-3 (EG-3) and Edgio-4 (EG-4), (b) Imperva-6 (IM-6), (c) Imperva-6 and its DNS global anycast network (IM-NS) after excluding non-overlapping sites and peering ASes.***

authoritative name servers using global anycast at the same sites where they deploy hosting servers [13, 26, 62]. Using the anycast site enumeration method we describe in § 4.4, we find that all 48 sites we uncover for Imperva-6 overlap with the sites of its DNS global anycast network. We refer to Imperva's DNS global anycast network as Imperva-NS. Edgio's CDN sites do not overlap significantly with its authoritative name server network so we exclude Edgio from this study.

Furthermore, we uncover the set of peering ASes to which a CDN announces both its regional IP anycast prefixes and its DNS global IP anycast prefixes. Even if a CDN deploys its hosting servers and its authoritative domain name servers at the same site, it may announce its regional CDN IP anycast prefixes and its global DNS IP anycast prefixes to different peers with different policy configurations. Uncovering the common set of peering ASes enables us to emulate a global anycast network that shares the same sites and peers with a regional anycast CDN. We map the valid penultimate hop (p-hop) in each RIPE Atlas probe's traceroute to a regional anycast IP (or the global DNS anycast IP) to the AS or the Internet Exchange Point (IXP) that owns the p-hop's IP address. We use RouteViews' BGP archive [57] of the same day when we collect the probes' traceroutes and the published IP prefixes from PeeringDB [27] to construct the IP-to-AS or IP-to-IXP mapping. In 49.0% of the traceroutes, the p-hops' IPs belong to IXPs and are not visible in BGP. After this step, for each overlapping site between Imperva-6 and Imperva-NS, we construct the set of common peers (ASes or IXPs) at that site.

Moreover, we assume that Imperva does not apply different latency-impacting policies when it announces a regional anycast IP prefix or a DNS global anycast IP prefix to the same peer at the same site. We validate this assumption by comparing the RTTs from the same probe reaching the same site via a regional IP or via a DNS global anycast IP. We find that the RTT differences are negligible (as shown in Figure 8 in Appendix D).

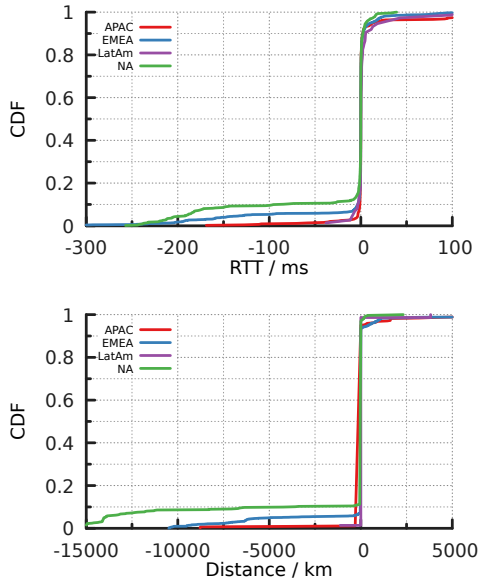


Figure 5: ***CDFs of RTT and distance differences between regional and global anycast.***

After completing these steps, we consider that the portion of Imperva-NS that includes the 48 overlapping sites with Imperva-6 and the overlapping set of peers with Imperva-6 at each site emulates the comparable global anycast counterpart of Imperva-6. This is because any site a probe reaches in the emulated global anycast network also announces a regional IP prefix to the same set of peers, and the RTT differences between reaching the site via a global anycast IP or a regional anycast IP are negligible.

Finally, to perform the comparison between regional anycast and global anycast, we instruct each RIPE Atlas probe to traceroute to its regional IP received via DNS when querying the representative hostname served by Imperva-6 and a global anycast IP of

Region (# probe groups)	$\Delta RTT < -5ms$ / $\lvert\Delta RTT\rvert \leq 5ms$ / $\Delta RTT > 5ms$	Closer Site	Same Site	Further Site
APAC (440)	15	26.7%	60.0%	13.3%
	395	0.0%	99.7%	0.3 %
	30	3.3%	33.3%	63.3 %
EMEA (2529)	219	69.9%	26.0%	4.1 %
	2130	1.0%	98.1%	0.8 %
	180	1.7%	22.8%	75.6%
LatAm (74)	5	0.0%	100.0%	0.0%
	60	0.0%	100.0%	0.0%
	9	11.1%	77.8%	11.1%
NA (584)	79	79.7%	19.0%	1.3%
	473	0.6%	97.9%	1.5%
	32	0.0%	68.8%	31.3%

Table 4: We examine the number of probe groups that have better/similar/worse RTTs (5 ms chosen as the threshold) in regional anycast than in global anycast. In each performance group, we examine the percentage of probe groups that reach closer, same, or further sites.

Imperva-NS, respectively. We filter the probes in the following cases: 1) probes that do not have a valid p-hop in their traceroute outputs; 2) probes that reach a non-overlapping site in Imperva-6 and Imperva-NS; and 3) probes that reach their catchment sites via a non-overlapping peer AS in Imperva-6 and Imperva-NS. In total, there are 4,417 probe groups with successfully resolved p-hops and we retain 82.1% (3,627 out of 4,417) of them after this filtering step.

Comparison of Imperva-6 and Imperva-NS Figure 4c plots the CDFs of the probe groups' RTTs to regional IPs in Imperva-6 and to global anycast IPs in Imperva-NS after excluding two networks' non-overlapping peers and sites, respectively. For probes in EMEA and NA, we see improved tail latency for regional anycast. For example, the 90th percentile RTT of probes in NA decreases from 110 ms to 38 ms. For probes in APAC and LatAm, regional anycast performs slightly worse than global anycast. Figure 5 plots the RTT difference and the distance difference between a probe's catchment site in Imperva-6 and in Imperva-NS. We observe that the percentage of probes with a distance reduction in regional anycast correlates well with the percentage of probes with latency reduction.

Further, we examine in detail the probes whose RTTs differ by 5 ms in Imperva-6 and Imperva-NS and how the RTT differences correlate with a probe's distance difference between its global anycast catchment site and its regional anycast catchment site. Table 4 summarizes the results. For the EMEA and NA regions, we find that when the probes achieve better performance in regional anycast, the majority of them reach closer sites in regional anycast than in global anycast. In the EMEA region, 69.9% (out of 219) probe groups that achieve more than 5 ms latency reduction reach closer sites in regional anycast. In the NA region, 79.7% (out of 79) probe groups that achieve more than 5ms latency reduction reach closer sites.

For the EMEA region, 7.1% probe groups (180 out of 2529) experience more than 5 ms longer latency in regional anycast than in global anycast; the majority (75.6%) of them reach further away sites. This result is consistent with our observation in § 5.1 that DNS mappings in the EMEA region are inefficient. Overall, there are more probe groups in EMEA that improve their performance in regional anycast. So we observe that the client latency distribution has improved in regional anycast. The probe groups with significant latency differences in other cases are few and we do not observe a consistent pattern.

For probe groups in each region that have similar performance in regional anycast and global anycast, between 97.9% to 100% of them reach the same sites. For probe groups in each region that have better or worse performance, some of them also reach the same sites. We examine the traceroute outputs of those probes and find that some of them reach the same sites via different AS paths and others have the same AS paths but different RTTs. BGP's route-selection uncertainty [4] and route instability (*e.g.*, temporary congestion) could potentially explain the RTT differences in these cases.

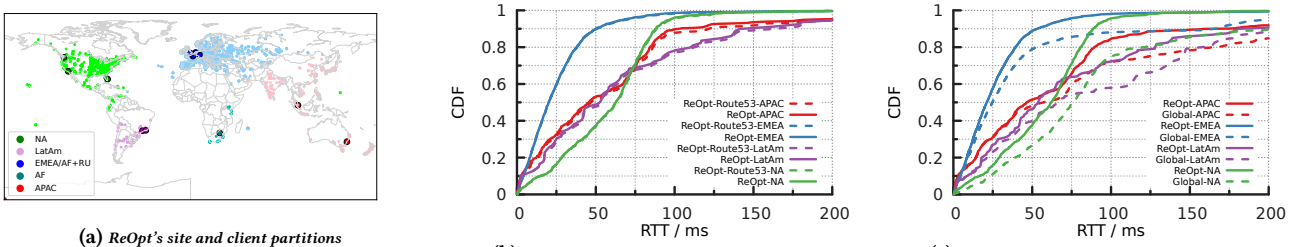
Takeaways: In the EMEA and NA regions, Imperva's regional anycast CDN effectively directs clients to nearby CDN sites and reduces client tail latency compared to global anycast. For example, the 90th percentile client latency is reduced by 72 ms for probe groups in NA, and the 95th percentile client latency is reduced by 98 ms and 165 ms for probes in EMEA and NA, respectively.

5.4 Case Studies: Causes of Latency Reduction

We have shown that regional anycast can reduce client latencies by restricting the CDN sites a client reaches to a specific geographic area. This result begets the question: why does the client not reach the same site in global anycast? To understand the reasons, we compare the traceroutes from probes to their catchment sites in regional anycast (Imperva-6) with those in global anycast (Imperva-NS). For probe groups with more than 5 ms latency reduction in regional anycast, we map each valid IP address in a traceroute output to the corresponding AS number with pyASN [30] and RouteViews BGP table dumps [57] archived on the same day we ran the traceroutes. We use the same method described in § 5.3 to identify IP addresses of IXPs. We also obtain AS relationships from CAIDA's AS relationship database [28]. We find two distinct cases where regional anycast can "override" common BGP policies to prevent clients from reaching distant CDN sites.

Overriding AS Relationship Preferences A common BGP policy is to prefer customer routes to peer routes and prefer peer routes to provider routes. As we show in Figure 1, this routing policy can cause probes to reach remote CDN sites within a large provider's customer cones instead of nearby CDN sites connected to the large provider's peers. Such situations frequently occur when a global IP anycast prefix from a CDN is announced to a non-tier-1 provider and subsequently announced to a tier-1 provider. Similar scenarios also happen when a client network peers with a large ISP that receives a global IP anycast prefix from a remote CDN site, but the client network also has a transit service provider that connects to a nearby regional anycast site. With regional anycast, clients will not receive the regional IPs advertised by those sites preferred more by BGP from DNS, thereby reaching closer sites and achieving lower latencies. In total, we observe that overriding this BGP policy accounts for 44.1% of the cases where regional anycast reduces client latencies in Imperva-6.

Overriding Peering Type Preferences Another common BGP policy is to prefer public peers to route server peers at an Internet Exchange Point (IXP) [60]. Public peers are ASes that directly exchange route advertisements and traffic over the fabric of an



(a) *ReOpt's site and client partitions*

(b) *Regional w/ and w/o DNS configuration*

(c) *Regional w/ ReOpt partitions vs. Global*

Figure 6: ***(a) Large colored dots show where Tangled sites are located. Small dots show where the probes are. Probes are assigned regional IPs announced by the same-color sites. (b) CDFs of probe RTTs between Route 53 DNS mapping and direct probe-to-regional-IP assignment. (c) Regional vs. global performance comparison using the Tangled testbed and the ReOpt client and region partition scheme.***

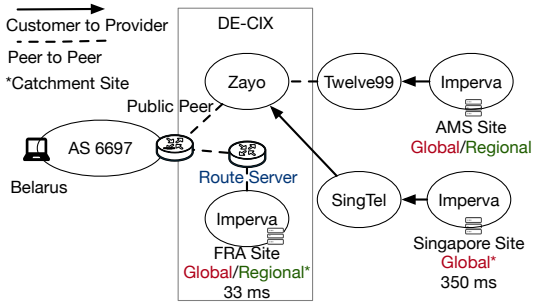


Figure 7: ***With a global anycast configuration, the Belarusian probe in AS 6697 reaches the CDN site in Singapore because public peering is more preferred to route server peering; with regional anycast, the probe reaches the site in Germany.***

IXP. In contrast, route server peers are ASes that exchange route advertisements via an intermediary route server [1]. Routers generally prefer public peers over route server peers, as public peering typically offers better performance than route server peering [60].

This BGP policy can also cause probes to reach remote CDN sites connected to public peers instead of nearby CDN sites connected to route server peers. We show an example in Figure 7. AS 6697 has a public peering with Zayo and a router server peering with Imperva at DE-CIX [21]. With global anycast, AS 6697 prefers the route to the global IP anycast prefix advertised by Zayo to the route advertised by Imperva at the FRA site because Zayo is a public peer and Imperva is a route server peer. However, Zayo prefers the route advertised by SingTel, as it is a customer route. Consequently, the probe located in Belarus in AS 6697 reaches the Singapore site in the global anycast. In contrast, in regional anycast, the probe does not receive the regional IPs of the Singapore site and hence reaches the closer FRA site.

Using IXPs' published route server feeds and CAIDA's AS relationship database [28], we identify that 1.6% of the cases where regional anycast reduces client latencies fall into this category. This number is small because many IXPs do not publish route server feeds so we cannot infer two ASes' peering type. For the rest of the cases where regional anycast improves performance, we are not able to clearly identify the reasons. This result could be due to missing IP hops in traceroutes, imperfect AS or AS relationship inferences, or unknown BGP policies.

6 Potentials

The measurement study in previous sections reveals the impact of DNS mapping inefficiencies on regional anycast's performance. This naturally leads us to question how much performance improvement regional anycast can achieve over global anycast if we improve its DNS mapping strategies. In this section, we explore this question using the Tangled testbed.

6.1 Latency-based Regional Partition

We develop a region partition and client mapping strategy, referred to as ReOpt, to address DNS mapping inefficiencies. First, we partition the Tangled testbed into geographic regions using the K-Means algorithm [46]. This step partitions geographically-close sites into regions. Second, we measure the unicast latency from each probe to each Tangled site and assign the probe to the region where its lowest unicast latency site is. Finally, we generate the country-level client-to-region mapping. For each country, we map all probes in the same country to the region where the majority of the country's probes are assigned to. This final step enables a network operator to use country-level IP geolocation to map a client to a regional IP.

To obtain the optimal number of regions, we vary the number of regions from three to six and calculate the average client latency under each regional partition. We find that a 5-region partition has the lowest average client latency on the Tangled testbed. Figure 6a shows the site partition and the client mapping of ReOpt's regional anycast configuration.

We observe two main differences between ReOpt's regional partitions and the regional partitions used by Edgio and Imperva. First, there is a separate region in Africa in ReOpt, while that region is part of the EMEA region in both Edgio and Imperva's regional partitions. Second, some probes in Central America are separated from probes in South America and are mapped to the North America region, but they are grouped together with the probes in South America in Edgio-4 and Imperva-6.

6.2 Global vs. Regional on Tangled

Next, we configure the Tangled testbed to deploy both global IP anycast and regional IP anycast and study their performance differences. When deploying global IP anycast, we configure all 12 sites in the Tangled testbed to announce one IP prefix to all their peering ASes and measure the RTTs from RIPE Atlas. When deploying regional anycast, we configure the sites in each of the five regions, as

determined by the ReOpt algorithm, to announce a distinct regional IP prefix to all their peering ASes.

We then conduct two regional anycast experiments. In the first experiment, we directly assign each probe a regional IP that contains its lowest-unicast-latency site and measure the RTT to each probe's assigned regional IP. We use this step to study the optimal regional anycast performance without IP geo-location errors and without aggregating probes by country.

In the second experiment, we use a commercial DNS provider, Amazon Route 53 [3], to configure a country-level client-to-regional-IP mapping. Route 53 supports both country-level and continent-level DNS mappings. We create a test domain name and delegate it to Route 53 and use Route 53's country-level mapping configuration tool to map clients from a country to a regional IP based on the mapping generated by the ReOpt algorithm. We then instruct the RIPE Atlas probes to ping the test domain name and measure their RTTs.

Figure 6b shows the CDFs of the probes' RTTs when we directly assign a probe to a regional IP and the probes' RTTs when we use Route 53. We can see that the performance of regional anycast is similar under these two configurations, while Route 53 mapping causes a slight performance degradation in the APAC and SA regions. This result suggests that IP geolocation errors in the country-level DNS mapping service provided by commercial DNS providers like Route 53 have a negligible adverse impact on implementing regional anycast.

Then we compare the performance of regional anycast with that of global anycast on the Tangled testbed. Figure 6c compares the CDF of the probes' RTTs under ReOpt's regional anycast configuration with Route 53 with that under the global anycast configuration. We observe that in the regional IP anycast configuration, the client latency in all regions has improved compared to the global anycast. For instance, ReOpt's five-region configuration shortens the 90th percentile latency for the probe groups in NA from 232.6 ms to 88.6 ms. This result shows that with latency-based region partition and client mapping, regional anycast can outperform global anycast, even in the presence of DNS mapping inefficiencies.

We note that latency-based regional partition and client mapping requires that a CDN operator measures the client latency distribution to its regional IPs. This requirement increases the design complexity of regional anycast. However, managing client-to-regional-IP mapping is still simpler than managing client-to-site mapping, as done in DNS-based CDNs, because a regional IP covers multiple sites and an operator need not manage load-balancing and fault tolerance among those sites.

7 Related Work

There exists a large body of work measuring the IP-layer anycast adoption in the Internet [14] or characterizing the performance of global IP anycast systems, including root DNS servers [7, 8, 16, 19, 25, 39, 40, 42, 43, 58, 68] and global anycast CDNs [11, 16, 39].

The blog article [47] discusses why LinkedIn adopted regional anycast for its private CDN and it reports that the client latency improved after the CDN switched from global anycast to regional anycast.

Differently, this work characterizes the deployments and studies in-depth the performance of two global-scale public regional anycast CDNs. We experimentally validate regional anycast's performance advantages and uncover its drawbacks. In addition, we use the Tangled testbed to explore how to improve the performance of regional anycast.

The performance challenges of global anycast systems motivated a few proposals to improve its performance [8, 22, 26, 42, 49, 61, 69]. Among the existing proposals, we consider regional anycast as the most promising approach as we discuss in § 2. Therefore, we aim to understand the deployments and performance of two real-world regional IP anycast CDNs in this work. We leave a comparison between regional anycast and other proposals as future work.

This work builds on McQuistin *et al.*'s work [49] of using the penultimate hops in traceroute outputs to infer the number of AS-level connections an anycast network has. Differently, in this work, we combine multiple sources, including rDNS, penultimate hops, IP-geolocation databases, RTT ranges, and RIPE Atlas probe locations to enumerate the CDN sites that announce an anycast IP prefix. iGreedy [15] identifies an IP anycast prefix and geolocates the anycast instances using iterative latency measurements from known vantage points. We experimented with iGreedy for anycast site enumeration and found that it mapped fewer published CDN sites than the method we used in this work.

8 Conclusion

Regional anycast, combining IP anycast and DNS redirection, has been deployed in practice. Yet how well it performs and how a CDN deploys it remain unknown. In this paper, we perform a comprehensive study to characterize the deployments and performance of two real-world regional IP anycast CDNs. In particular, we explore the region division strategies of the CDNs and how they map clients to regions. We find that the CDNs divide their networks into regions by continents or large countries and similarly, the CDNs assign clients to regional IPs by the continents or the countries they reside in. Our measurements show that regional IP anycast in general can mitigate the catchment inefficiency problem experienced by global IP anycast, but poor DNS mapping, where DNS returns regional IPs originating from sub-optimal CDN sites to clients, could worsen client latencies. Using the Tangled testbed, we show that with a latency-based region partition method, regional anycast can reduce client tail latencies in various geographical areas, overcoming DNS mapping inefficiencies. Based on these results, we conclude that deploying regional anycast is worthwhile, as it can effectively override BGP's sub-optimal route selections by upper-bounding the distance between clients and their CDN catchment sites.

Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province (2020B0101390001), the National Science Foundation under Award 2225448, China NSF grants (62172206), and an Internet Freedom Fund from the Open Technology Fund (OTF). We gratefully thank Italo Cunha, Ethan Katz-Bassett, Jiangchen Zhu, Leandro Bertholdo, and Raffaele Sommese for helping us with experiments on the PEERING and Tangled testbeds.

References

[1] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. Anatomy of a Large European IXP. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'12)*, 2012.

[2] Akamai Technologies. Akamai Online Retail Performance Report: Milliseconds are Critical. Retrieved in Sep, 2022 from https://www.ir.akamai.com/news-releases/news-release-details/akamai-online-retail-performance-report-milliseconds-are.

[3] Amazon. Values Specific for Geolocation Records. Retrieved in Sep, 2022 from https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-values-geo.html#rrsets-values-geo-location.

[4] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Ítalo Cunha, Phillipa Gill, and Ethan Katz-Bassett. Investigating Interdomain Routing Policies in the Wild. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'15)*, 2015.

[5] Todd Arnold, Ege Gürmeriçliler, Georgia Essig, Arpit Gupta, Matt Calder, Vasileios Giotsas, and Ethan Katz-Bassett. (How Much) Does a Private WAN Improve Cloud Performance? In *Proceedings of the Annual Conference of the IEEE International Conference on Computer Communications (INFOCOM'20)*, 2020.

[6] Karthik Balakrishnan. Simplify Traffic Steering with Cloud DNS Routing Policies. Retrieved in Sep, 2022 from https://cloud.google.com/blog/products/networking/dns-routing-policies-for-geo-location--weighted-round-robin.

[7] Hitesh Ballani and Paul Francis. Towards a Global IP Anycast Service. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'05)*, 2005.

[8] Hitesh Ballani, Paul Francis, and Sylvia Ratnasamy. A Measurement-Based Deployment Proposal for IP Anycast. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'06)*, 2006.

[9] Leandro M Bertholdo, Joao M Ceron, Wouter B de Vries, Ricardo de Oliveira Schmidt, Lisandro Zambenedetti Granville, Roland van Rijswijk-Deij, and Aiko Pras. Tangled: A Cooperative Anycast Testbed. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM'21)*, 2021.

[10] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. Mapping the Expansion of Google's Serving Infrastructure. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'13)*, 2013.

[11] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitu Padhye. Analyzing the Performance of an Anycast CDN. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'15)*, 2015.

[12] Matt Calder, Manuel Schroder, Ryan Gao, Ryan Stewart, Jitu Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System. In *Proceedings of 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, 2018.

[13] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-User Mapping: Next Generation Request Routing for Content Delivery. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'15)*, 2015.

[14] Danilo Cicalese, Jordan Augé, Diana Joumblatt, Timur Friedman, and Dario Rossi. Characterizing IPv4 Anycast Adoption and Deployment. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '15)*, 2015.

[15] Danilo Cicalese, Diana Joumblatt, Dario Rossi, Marc-Olivier Buob, Jordan Augé, and Timur Friedman. A Fistful of Pings: Accurate and Lightweight Anycast Enumeration and Geolocation. In *Proceedings of the Annual Conference of the IEEE International Conference on Computer Communications (INFOCOM'15)*, 2015.

[16] Danilo Cicalese and Dario Rossi. A Longitudinal Study of IP Anycast. *ACM SIGCOMM Computer Communication Review*, 48(1):10–18, 2018.

[17] Google Cloud. Cloud CDN. Retrieved in January, 2023 from https://cloud.google.com/cdn.

[18] Cloudflare. Cloudflare CDN Reference Architecture. https://cf-assets.www.cloudflare.com/slt3lc6tev37/18dA4NLfq8oXY8EVZxPlpY/b9cab82be79ebefa80f08c09eaa3d93e/Cloudflare_CDN_Reference_Architecture.pdf, 2022.

[19] Lorenzo Colitti, Erik Romijn, Henk Uijterwaal, and Andrei Robachevsky. Evaluating the Effects of Anycast on DNS Root Name Servers. Retrieved in Sep, 2022 from https://www.ripe.net/publications/docs/ripe-393.

[20] Carlo Contavalli, Wilmer van der Gaast, David C Lawrence, and Warren "Ace" Kumari. Client Subnet in DNS Queries. Technical report, RFC Editor, Dec 2016.

[21] DE-CIX. DE-CIX GlobePEER Looking Glass. Retrieved in Feb, 2023 from https://lg.de-cix.net/.

[22] Ricardo de O. Schmidt, John Heidemann, and Jan Kuipers. Anycast Latency: How Many Sites Are Enough? In *Proceedings of Passive and Active Measurement (PAM'17)*, 2017.

[23] Wouter B De Vries, Roland van Rijswijk-Deij, Pieter-Tjerk De Boer, and Aiko Pras. Passive Observations of a Large DNS Service: 2.5 Years in the Life of Google. *IEEE transactions on network and service management*, 17(1):190–200, 2019.

[24] EdgeCast. EdgeCast POPs. Retrieved in Sep, 2022 from https://docs.edgecast.com/cdn/Content/Reference/POP_Listing.htm.

[25] Xun Fan, John Heidemann, and Ramesh Govindan. Evaluating Anycast in the Domain Name System. In *Proceedings of the Annual Conference of the IEEE International Conference on Computer Communications (INFOCOM'13)*, 2013.

[26] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs. In *Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI'15)*, 2015.

[27] Center for Applied Internet Data Analysis (CAIDA). CAIDA Data Server-PeeringDB Archive Aug,30,2022 Data. Retrieved in Feb, 2023 from https://publicdata.caida.org/datasets/peeringdb/2022/08/peeringdb_2_dump_2022_08_30.json.

[28] Center for Applied Internet Data Analysis (CAIDA). The CAIDA AS Relationships Dataset, Aug,30,2022 Data. Retrieved in Feb, 2023 from https://www.caida.org/catalog/datasets/as-relationships/.

[29] Manaf Gharaibeh, Anant Shah, Bradley Huffaker, Han Zhang, Roya Ensafi, and Christos Papadopoulos. A Look at Router Geolocation in Public and Commercial Databases. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'17)*, 2017.

[30] Hadi Asghari. Pyasn · PyPI. Retrieved in Feb, 2023 from https://pypi.org/project/pyasn/.

[31] Shuai Hao, Yubao Zhang, Haining Wang, and Angelos Stavrou. End-Users Get Maneuvered: Empirical Analysis of Redirection Hijacking in Content Delivery Networks. In *Proceedings of 27th USENIX Security Symposium (USENIX Security'18)*, 2018.

[32] Paul Hoffman, Andrew Sullivan, and Kazunori Fujiwara. DNS Terminology. RFC 8499, RFC Editor, Jan 2019.

[33] Bradley Huffaker, Marina Fomenkov, and Kc Claffy. Geocompare: A Comparison of Public and Commercial Geolocation Databases. Technical report, Cooperative Association For Internet Data Analysis (CAIDA), 2011.

[34] The International Air Transport Association (IATA). IATA Airport Code. Retrieved in Sep, 2022 from https://www.iata.org/en/publications/directories/code-search/.

[35] iconectiv. CLLI code. Retrieved in Sep, 2022 from https://www.commonlanguage.com/resources/commonlang/productshowroom/product/clli/.

[36] Imperva. Imperva PoPs. Retrieved in Sep, 2022 from https://status.imperva.com/?_ga=2.95445268.988020403.1637323716-57897548.1602317115.

[37] IPInfo. IPinfo. Retrieved in Sep, 2022 from https://ipinfo.io/.

[38] Weifan Jiang, Tao Luo, Thomas Koch, Yunfan Zhang, Ethan Katz-Bassett, and Matt Calder. Towards Identifying Networks with Internet Clients Using Public Data. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'21)*, 2021.

[39] Thomas Koch, Ke Li, Calvin Ardi, Ethan Katz-Bassett, Matt Calder, and John Heidemann. Anycast in Context: A Tale of Two Systems. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'21)*, 2021.

[40] JH Kuipers. Analysing the K-root Anycast Infrastructure. Retrieved in Sep, 2022 from https://labs.ripe.net/Members/jh_kuipers/analyzing-the-k-root-anycast-infrastructure.

[41] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*, 2019.

[42] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Internet Anycast: Performance, Problems, & Potential. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'18)*, 2018.

[43] Ziqian Liu, Bradley Huffaker, Marina Fomenkov, Nevil Brownlee, and Kimberly Claffy. Two Days in the Life of the DNS Anycast Root Servers. In *Proceedings of Passive and Active Measurement (PAM'07)*, 2007.

[44] Matthew Luckie, Bradley Huffaker, and K Claffy. Learning Regexes to Extract Router Names from Hostnames. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'19)*, 2019.

[45] Matthew Luckie, Bradley Huffaker, Alexander Marder, Zachary Bischof, Marianne Fletcher, and K Claffy. Learning to Extract Geographic Information from Internet Router Hostnames. In *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT'21)*, 2021.

[46] J MacQueen. Classification and Analysis of Multivariate Observations. In *Proceedings of the 5th Berkeley Symp. Math. Statist. Probability*, 1967.

[47] Ritesh Maheshwari. TCP over IP Anycast - Pipe Dream or Reality? Retrieved in Sep, 2022 from https://engineering.linkedin.com/network-performance/tcp-over-ip-anycast-pipe-dream-or-reality, 2015.

[48] MaxMind Inc. MaxMind. Retrieved in Sep, 2022 from https://www.maxmind.com/en/home.

[49] Stephen McQuistin, Sree Priyanka Uppu, and Marcel Flores. Taming Anycast in the Wild Internet. In *Proceedings of the Annual Conference of the ACM Internet Measurement Conference (IMC'19)*, 2019.
[50] Trevor Mendez, Walter Milliken, and Dr. Craig Partridge. Host Anycasting Service. RFC, RFC Editor, Nov 1993.
[51] Microsoft. What is a Content Delivery Network on Azure? Retrieved in January, 2023 from https://docs.microsoft.com/en-us/azure/cdn/cdn-overview.
[52] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. Pruning Edge Research with Latency Shears. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, 2020.
[53] Aaron Peters. CDN Finder (CDN Lookup Made Easy). Retrieved in Sep, 2022 from https://www.cdnplanet.com/tools/cdnfinder/.
[54] Aaron Peters. CDNPlanet - EDNS Client Subnet Checker. Retrieved in Sep, 2022 from https://www.cdnplanet.com/tools/edns-client-subnet-checker/.
[55] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. IP Geolocation Databases: Unreliable? *ACM SIGCOMM Comput. Commun. Rev.*, 41(2):53–56, 2011.
[56] Yakov Rekhter and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, RFC Editor, Jan 1994.
[57] RouteView. BGP RIB Archive Aug,30,2022 Data. Retrieved in Feb, 2023 from ftp://archive.routeviews.org//bgpdata/2022.08/RIBS/rib.20220830.0000.bz2.
[58] Sandeep Sarat, Vasileios Pappas, and Andreas Terzis. On the Use of Anycast in DNS. In *Proceedings of the 2005 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'06)*, 2006.
[59] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. PEERING: Virtualizing BGP at the Edge for Research. In *Proceedings of the 15th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '19)*, 2019.
[60] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'17)*, 2017.
[61] Kyle Schomp and Rami Al-Dalky. Partitioning the Internet Using Anycast Catchments. *ACM SIGCOMM Comput. Commun. Rev.*, 50(4):3–9, 2020.
[62] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K. Sitaraman. Akamai DNS: Providing Authoritative Answers to the World's Queries. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'20)*, 2020.
[63] Yuval Shavitt and Noa Zilberman. A Geolocation Databases Study. *IEEE Journal on Selected Areas in Communications*, 29(10):2044–2056, 2011.
[64] Neil Spring, Ratul Mahajan, and Thomas Anderson. The Causes of Path Inflation. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'03)*, 2003.
[65] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'02)*, 2002.
[66] RIPE NCC Staff. RIPE Atlas: A Global Internet Measurement Network. *Internet Protocol Journal*, 18, 2015.
[67] Akamai Technologies. EdgeScape. Retrieved in Sep, 2022 from https://developer.akamai.com/edgescape.
[68] Lan Wei and John Heidemann. Does Anycast Hang Up on You? In *Proceedings of the Network Traffic Measurement and Analysis Conference (TMA'17)*, 2017.
[69] Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M. Maggs, Haiying Shen, Ramesh K. Sitaraman, and Xiaowei Yang. AnyOpt: Predicting and Optimizing IP Anycast Performance. In *Proceedings of the Annual Conference of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'21)*, 2021.

Appendix

Appendices are supporting material that has not been peer-reviewed.

A Collection of CDN's Technical documents

Existing work [31, 54] has revealed that Edgio and Imperva are the two CDNs deploying regional anycast. To validate this observation and obtain a more recent view, we manually collect and examine the official technical documents to identify the studied CDNs' redirection mechanisms. As illustrated in Section 4.1, we obtain a list of the top 15 CDN providers by retrieving CDNFinder's API [53]. We note that we here focus on the CDN platform of each provider while global anycast could be available with other components to support specific services, *e.g.*, DNS or Cloud hosting. We also eliminate two other CDNs from our top list: (1) Facebook CDN, because it is deployed as a private CDN that supports Meta/Facebook's services; (2) Automatic CDN, which works as a plugin to accelerate the static assets of sites that are tied to Wordpress.com.

B IP Geolocating p-hops

We determine the catchment site of a RIPE Atlas probe by IP-geolocate the penultimate hop (p-hop) from its traceroute output to an IP anycast address. We then use the p-hop's geo-location to locate a CDN site. If the CDN site has an on-site router, the p-hop is often the site router so that we can observe a co-located CDN site and infer that the CDN site announces the regional IP address we traceroute to. If a CDN site does not have a site router and connects to a remote IXP via a link-layer connection [1], we will not observe a CDN site co-located with a p-hop. In this case, we assume that the CDN site closest to the p-hop announces the regional IP we traceroute to.

Because IP-geolocation at the city-level is not reliable [33, 55, 63], we combine a number of sources, including IP-geolocation databases, reserve DNS (rDNS) records, and RTT ranges to infer a p-hop's location. We use three IP-geolocation databases: MaxMind [48], ipinfo [37], and EdgeScape [67] to provide the possible locations of the p-hop. And we describe this process as follows.

Reverse DNS (rDNS): We first infer the location of a p-hop from its rDNS name [44, 45]. If a p-hop has an rDNS name and the name contains a geo-hint at the city level (*e.g.*, operator-defined codes, IATA/ICAO codes [34], or CLLI code [35]), we use the geo-hint to map the p-hop to a PoP location published by Edgio or Imperva. For instance, an rDNS name ae-65.core1.amb.edgecastcdn.net. indicates a p-hop is located in Amsterdam, Netherlands.

If the rDNS name does not contain any valid geo-hints at the city level, we will check the domain name's country code Top-Level Domain (ccTLD). If the CDN provider (Edgio or Imperva) has only one anycast site deployed in the ccTLD's country, we will map the p-hop to that site's city-level location.

RTT range: If a p-hop does not have an rDNS name or we could not infer its location from the rDNS name, we then attempt to use the location of the RIPE Atlas probe that went through the p-hop [29] and has less than 1.5 ms RTT to the p-hop to infer its location. The threshold is chosen according to the typical size of a metropolitan area, as the speed-of-light latency in fiber is roughly 100 km per 1 ms RTT. First, we query three IP-geo databases to estimate the p-hop's location, which may return different results. Then, we filter the invalid results based on the speed-of-light distance between the p-hop and the RIPE Atlas probe, and use the valid location closest to the RIPE Atlas probe as the p-hop's location. By this method, the location of the p-hop can be resolved when both the position of the probe and the IP-geo databases are correct at the same time.

Country-level IPGeo: The two steps above could resolve the majority of Imperva and Edgio's p-hops' locations. If after the above two steps, the location of the p-hop is still unresolved, we will use the country information from the IP-geolocation databases to infer the location of the p-hop. If all IP-geolocation databases return the same country location for the p-hop, and the CDN provider only lists one site in the country, we will use that listed site location as

Table 5: Top CDNs and their documents to demonstrate the types of redirection.

CDNs	Redirection Method	Documents
Google Cloud CDN	Global Anycast	https://cloud.google.com/cdn
Cloudflare	Global Anycast	https://www.cloudflare.com/network/
Amazon Cloudfront	DNS	https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/HowCloudFrontWorks.html
Akamai	DNS	https://www.akamai.com/our-thinking/cdn/what-is-a-cdn
Fastly	DNS & Global Anycast	https://docs.fastly.com/en/guides/using-fastly-with-apex-domains
Stackpath	Global Anycast	https://www.stackpath.com/edge-academy/what-is-anycast
Edgio (EdgeCast)	**Regional Anycast**	https://docs.edgecast.com/cdn/Content/HTTP_and_HTTPS_Data_Delivery/How_Do_Our_HTTP-Based_Platforms_Work.htm
bunny.net	DNS	https://bunny.net/network/smartedge
Alibaba Cloud	DNS	https://www.alibabacloud.com/help/en/alibaba-cloud-cdn
Imperva (Incapsula)	**Regional Anycast**	https://www.imperva.com/learn/performance/route-optimization-anycast/
Microsoft Azure*	Global Anycast	https://docs.microsoft.com/en-us/azure/cdn/cdn-overview
ChinanetCenter/Wangsu	DNS	https://en.wangsu.com/product/9
CDN77	DNS	https://www.cdn77.com/network
Tencent Cloud	DNS	https://intl.cloud.tencent.com/products/cdn
Vercel	DNS	https://vercel.com/docs/concepts/edge-network/regions

* Azure provides an Azure Route Server that can support multi-region anycast deployment similar to regional anycast, but it essentially works by leveraging the front load balancer and only uses for private networks over cloud infrastructure (https://docs.microsoft.com/en-us/azure/route-server/anycast).

Percentile	Imperva-6				Edgio-3				Edgio-4			
	APAC	EMEA	NA	LatAm	APAC	EMEA	NA	LatAm	APAC	EMEA	NA	LatAm
50-th	10 (10)	14 (12)	11 (14)	23 (22)	13 (12)	12 (12)	12 (12)	110 (110)	12 (12)	12 (11)	12 (12)	33 (25)
90-th	66 (90)	48 (43)	33 (35)	109 (109)	76 (61)	40 (41)	31 (33)	145 (144)	75 (62)	39 (41)	31 (32)	112 (110)
95-th	103 (124)	65 (62)	50 (49)	153 (162)	121 (121)	63 (73)	45 (45)	154 (148)	115 (121)	62 (71)	47 (44)	135 (126)

Table 6: Latency comparison between selected hostnames and other hostnames. RTTs are in the unit of milliseconds. Numbers in parentheses are RTTs of the aggregated results of the other hostnames. Tail latencies of Imperva-6 in EMEA and NA are similar.

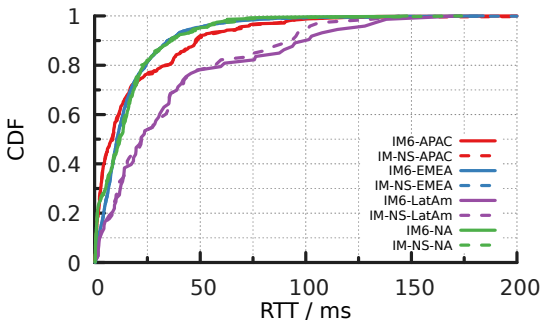


Figure 8: ***CDFs of the RTTs of the probes that reach the same CDN site via a regional IP anycast address and a global IP anycast address in Imperva-6 and Imperva-NS, after excluding the non-overlapping sites and peering ASes of the two networks.***

the p-hop's location, this is an effective way to discover the single site in one country.

Unresolved: We find that for the three sets of hostnames (Edgio-3, Edgio-4, and Imperva-6), we cannot resolve the locations of the p-hops of 2.3% to 9.9% valid traces. Increasing the RTT threshold will lead to an inaccurate inference. For those unresolved p-hops, even if we use IP geo-location databases to approximate their locations, we do not find more CDN sites. Therefore, we leave those p-hops unmapped.

C General performance

To ensure that the performance of the selected hostnames is representative, we conduct 12 additional measurements to random hostnames for each regional configuration and aggregate the results in Table 6. As a comparison, the performance of the representative hostnames is similar to that of the other hostnames, suggesting that the performance results of regional anycast are generalizable to other Edgio and Imperva hostnames.

D Same-Site Latency Comparison

When we study the performance of Imperva-6, we compare it with Imperva's DNS global anycast network (Imperva-NS) after excluding the non-overlapping sites and peers of the two networks. We assume that when Imperva announces a regional IP anycast prefix and a DNS global IP anycast prefix at the same site to the same set of peers, it does not apply different latency-impact policies to these prefixes. To validate this assumption, we compare the RTT of a RIPE Atlas probe that reaches the same site via a regional IP anycast address with the RTT of the probe that reaches the site via a global IP anycast address. We include only the results from the probes that reach the CDN sites via the common set of peering ASes observed in Imperva-6 and Imperva-NS.

Figure 8 shows the result. The differences in the RTT distributions are negligible, which indirectly validate that Imperva does not apply different latency-impacting policies to its regional IP anycast prefixes and its DNS global IP anycast prefixes. Because if it does, we should observe significantly different RTT distributions.